Software Simulation of an Unmanned Vehicle Performing Relative Spacecraft Orbits

Christopher C. Romanelli

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Aerospace Engineering

Hanspeter Schaub, Chair

Chris Hall

Craig Woolsey

April 27, 2006

Blacksburg, Virginia

Keywords: Dynamic Simulation, UMBRA, Dynamics and Control, Orbit Simulation

Christopher C. Romanelli

(ABSTRACT)

The challenge of sensing relative motion between vehicles is an important subject in the engineering field in recent years. The associated applications range from spacecraft rendezvous and docking to autonomous ground vehicle operations. The focus of this thesis is to develop the simulation tools to examine this problem in the laboratory environment. More specifically, the goal is to create a virtual unmanned ground vehicle that operates in the same manner as an actual vehicle. This simulated vehicle allows for safely testing other software or hardware components before application to the actual vehicle. In addition, the simulated vehicle, in contrast to the real vehicle, is able to operate on different surfaces or even different planets, with different gravitational accelerations. To accomplish this goal, the equations of motion of a two-wheel driven unmanned vehicle are developed analytically. To study the spacecraft application, the equations of motion for a spacecraft cluster are also developed. These two simulations are implemented in a modular form using the UMBRA framework. In addition, an interface between these two simulations is created for the unmanned vehicle to mimic the translational motion of a spacecraft's relative orbit. Finally, some of the limitations and future improvements of the existing simulations are presented.

This work received support from U.S. Defense Advanced Research Project Agency (DARPA) and Sandia National Laboratories.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sensing relative motion between vehicles is a challenge that applies to vehicles of all types. Some examples include spacecraft formations, aerial vehicles flying in close proximity, or even a formation of ground vehicles. The growing number of applications for this problem occur in fields such as the military, local police forces, and various space administrations. The applications for spacecraft formations alone range from wide field-of-view interferometry, to forming virtual structures, to engaging in close proximity flying operations, as well as rendezvous and docking maneuvers. One specific aircraft application is the creation a fully autonomous aerial refueling aircraft. In another example, a group of unmanned vehicles must navigate a crowded urban environment by following a lead vehicle to an unknown location and complete an objective. However, the task of sensing relative motion in any of these applications is difficult and complex. For example, a spacecraft has six degrees of freedom and operates in a near vacuum environment at speeds on the order of kilometers per

second. In addition, there are disturbances such as solar radiation pressure and atmospheric drag that affect the spacecraft's orbit and attitude. Now consider the application where a spacecraft attempts to rendezvous and dock with another spacecraft or maintain a 10-meter separation distance. The spacecraft rendezvous and docking application is depicted in Figure 1.1[1]. In order to gain a better understanding of the relative motion concept, some

Figure 1.1: Spacecraft Rendezvous and Docking Maneuver

institutions and laboratories are developing hardware/simulation test beds. These test beds combine hardware components such as air bearing tables or robotic ground vehicles, with software components such as numerical vehicle simulations or spacecraft orbit models. The hardware components are designed to examine a simpler form of the problem. For example, an air bearing table simulates the motion of a spacecraft in its orbit plane with only one degree of rotational freedom and two degrees of translational freedom. In contrast, the numerical simulation component both complements the hardware component and extends

[1]http://science.ksc.nasa.gov/mirrors/images/images/pao/ASTP/10076433.jpg

its capability. For instance, an advantage of having a numerical vehicle simulation is that software, such as a new control law, is safely tested before applying it to the actual vehicle. However, the more important advantage of having a numerical vehicle simulator is that it is not limited to operating in a lab environment. For example, a simulated ground vehicle is able to operate on different types of surfaces, such as gravel or slippery pavement, and even different planets which have different gravitational and surface properties. An example application would be to model an unmanned vehicle on Mars as it is depicted in Figure 1.2[2]. This vehicle simulation capability is extended to make the actual vehicle *behave* as

Figure 1.2: Unmanned Vehicle on Mars

if it is operating on different surfaces or planets, or even to mimic the orbital motion of a spacecraft. Figure 1.3 depicts how this simulation-hardware interaction would occur with an autonomous ground vehicle. In this example, a software module is created to control both the real and virtual versions of the ground vehicle. This control module interacts with

[2]http://marsrovers.nasa.gov/gallery/artwork/hires/rover1.jpg

the actual and simulated vehicles in the same manner. In addition, a sensor or another simulation is able to send the same information to either type of vehicle, without having to make any distinction between the two. This example illustrates the importance of having an efficient hardware/simulation testbed to study the relative motion problem.

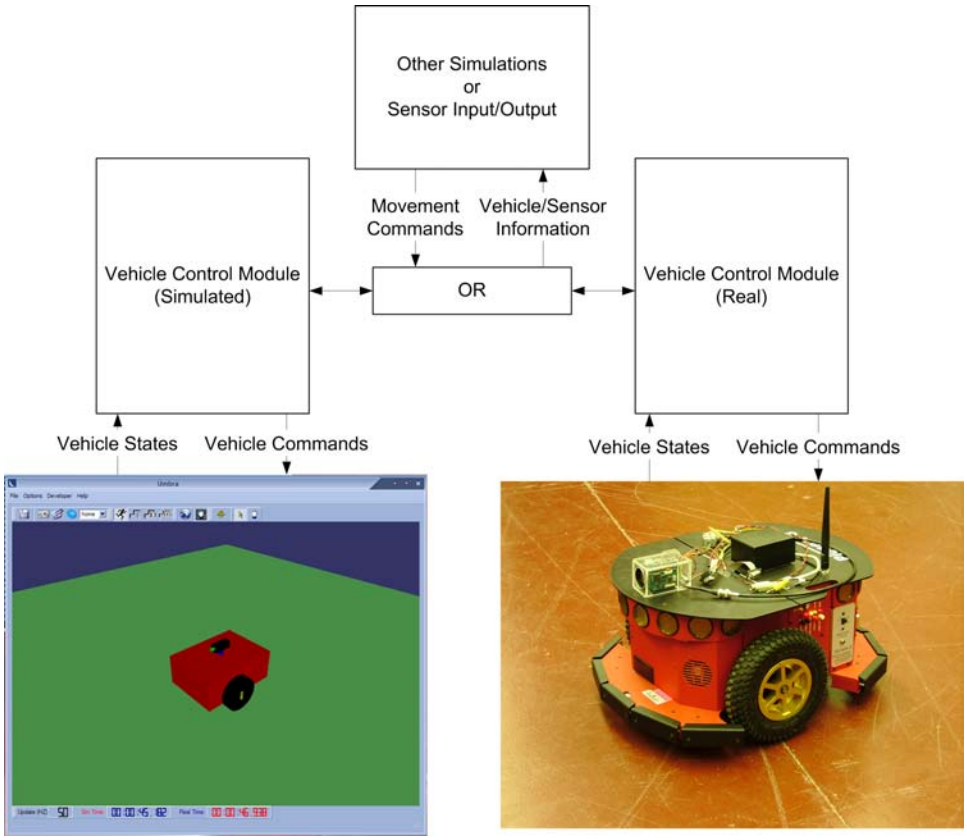


Figure 1.3: Depiction of Simulation/Hardware Interaction

1.1 Historical Studies of the Relative Motion Problem

The study of relative motion concepts for different types of vehicles is challenging to do in the laboratory environment. The challenge lies in trying to reproduce the types of environments these vehicles operate in. Historically, one application that has been studied in great detail is the motion of spacecraft, which includes the translational and attitude dynamics of spacecraft. With the spacecraft application the difficulties arise when trying to mimic a frictionless and nearly torque-free environment in a ground-based laboratory.[24] The solution that has frequently been used in past is an air-bearing environment. In an air-bearing environment, pressurized air is pumped through small orifices and provides a thin lubricating film. These air-bearing platforms are typically in a spherical or planar shape, which gives them the capability to study planar or attitude motion of spacecraft respectively.[24] Other types of systems that produce similar types of environments are water tanks or magnetic suspension systems. However, these hardware test beds have their limitations when applied to spacecraft. For instance, the magnetic suspension systems have a limited range of motion, and submerging a satellite is not a useful way to study its motion in space. For example, a water tank is limited in size and does not model any orbital dynamics.[24] In contrast, an autonomous ground vehicle operates indoors or outdoors and has the ability to simulate the translational motion of a spacecraft.

Of interest to the relative motion problem are the planar air-bearing test beds. These testbeds provide two degrees of translation freedom and one degree of rotational freedom. This

type of test bed provides a good environment for simulating and studying planar relative-motion of spacecraft. One specific application that is studied on these test-beds is rendezvous and docking.[24] In the 1970's these facilities were a common occurrence in the industry. They featured large polished floors with floating air-bearing vehicles.[12] Furthermore, NASA has manufactured planar air-bearing testbeds capable of supporting 200-pound vehicles that are either manned or unmanned.[10] Another notable facility that uses a planar air-bearing is the Aerospace Robotics Laboratory at Stanford University. This facility features a 6-by-12 foot polished granite table with a robotic arm and various vehicles that freely float on the table. This test bed is used to study the problem of capturing a free floating object with a robotic arm, which is pictured in Figure 1.4.[23] Other facilities that include similar systems are a

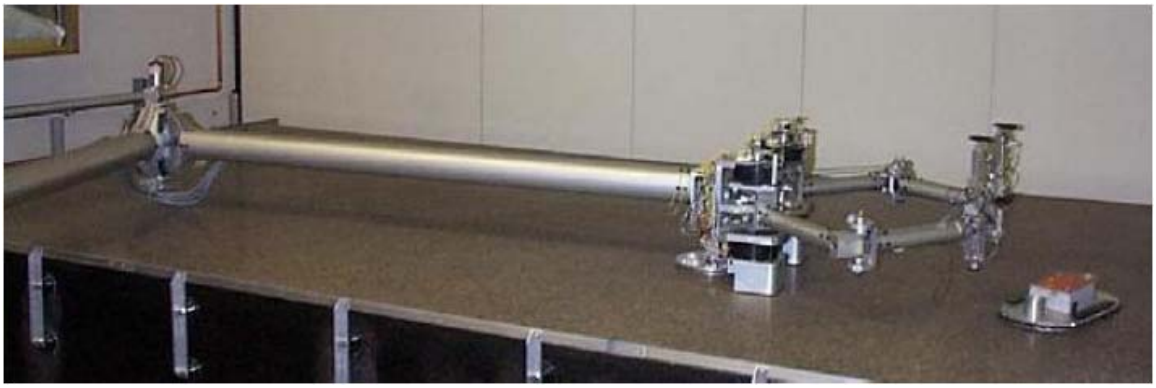
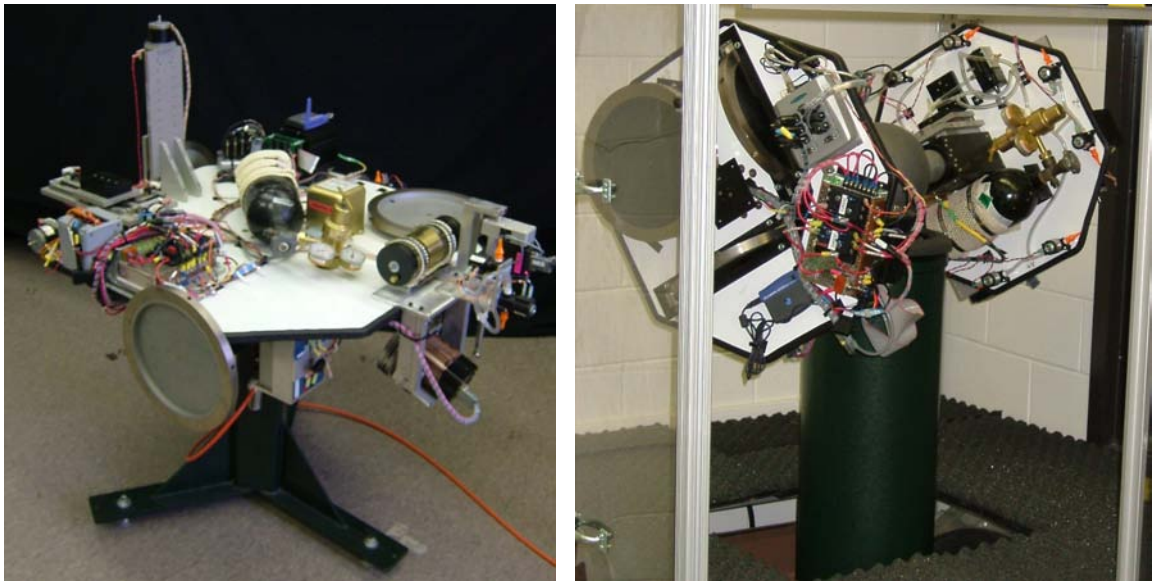


Figure 1.4: Planar Air-Bearing Testbed at Stanford's ARL[23]

10-by-16 foot plate glass table at the Tokyo Institute of Technology and a planar air-bearing at the University of Victoria.[4,16] The facilities mentioned here represent only a few of these planar air-bearing testbeds. Additionally, these air-bearing systems are limited by table-size and weight and do not model spacecraft orbital motion. In contrast, spacecraft attitude

and rotational motion is explored using spherical air bearings. Currently at Virginia Tech, the Space Systems Simulation Laboratory (SSSL) is exploring this problem using table-top and dumbbell style air bearing systems. The table-top system provides one full 360-degree of freedom, while the dumbbell system has two full rotational degrees of freedom.[1,15] The air-bearings are pictured in Figure 1.5. However, the spherical air-bearing systems only sim-



(a) "Tabletop" Style Air Bearing (b) "Dumbbell" Style Air Bearing

Figure 1.5: Spherical Air-Bearing Systems at the SSSL[1,15]

ulate the attitude motion of spacecraft and do not feature any of the orbital and attitude dynamics. For example, the gyroscopic coupling of attitude and orbital motion or the effect of gravity gradient torque. In addition, air-bearing systems are mathematically complex and expensive as well as having limited or no mobility. This is where autonomous vehicles have a clear advantage. An autonomous vehicle system is able to mimic not only the motion of spacecraft, but also aircraft and ground-based vehicles. With the increasing interest in visual

sensing techniques, the mobility of an autonomous system provides various different lighting environments to explore, as well as different types of surfaces and obstacles. In laboratories that feature autonomous ground vehicles, a hardware/simulation test bed is created in order to add software components to compliment the hardware. At the University of Florida, the Nonlinear Controls and Robotics (NCR) group (which is part Autonomous and Multi-Agent Systems (AMAS) Laboratory at UF) is examining the problem of using a visual servo to control autonomous robotic vehicles.[17,28] However, this facility is limited to exploring only control solutions for ground vehicles. At Virginia Tech, a new facility is being developed using autonomous robotic vehicles to explore many different types of relative motion problems, such as spacecraft orbital motion and aircraft motion.

1.2 The Autonomous Vehicle Systems Lab

Virginia Tech is developing a new facility called the Autonomous Vehicle Systems (AVS) Lab to examine the relative motion problem. The goal of this lab is to mimic the near-planar relative dynamics of ground, aerial or space-based vehicles and to provide relative motion sensors with realistic physical motion. This goal is achieved by developing a sophisticated hybrid hardware/simulation environment. For example, a software component that models spacecraft orbital motion interfaces with the vehicle's control software, thus enabling the actual vehicle to behave like a spacecraft. This feature is extended to mimic aircraft or even ground vehicles on complex terrains. The current hardware in the lab includes an

ActivMedia P3-DX robotic vehicle that is used to provide a 2D autonomous motion platform. Additional hardware includes two cameras to examine visual sensing techniques. In addition, these cameras are mounted on pan and tilt units which allow more flexibility to examine out-of-plane or lateral attitude motion of a vehicle. Currently ground vehicle relative control issues are being investigated using only visual sensing techniques. However, the applications for this vehicle can be expanded to mimic the relative motion of aircraft or spacecraft. For example, the autonomous vehicle mimics the actual in-plane motion of spacecraft. Another benefit of using autonomous vehicles is that the relative motion tests can be performed both indoors and outdoors, allowing for a diverse set of lighting and surface conditions to be tested. The simulation capability within in the lab is achieved through a

Figure 1.6: ActiveMedia Robotic Vehicle Pioneer with Pan-and-Tilt Unit

software framework called UMBRA and is being used to generate a modular code environment. Different vehicle tasks such as communication, control, sensing processing, etc., are encapsulated into compiled code objects called modules. These modules are then connected to each other, allowing for interaction and manipulation in real-time. This feature of the UMBRA framework allows us to develop modules that can easily interact with either the actual hardware or a simulated vehicle without having to recompile any code. Figure 1.7 shows an depiction of this simulation environment. Here the vehicle control components are put into modules and interfaced with an orbit simulator. By sharing this code through the

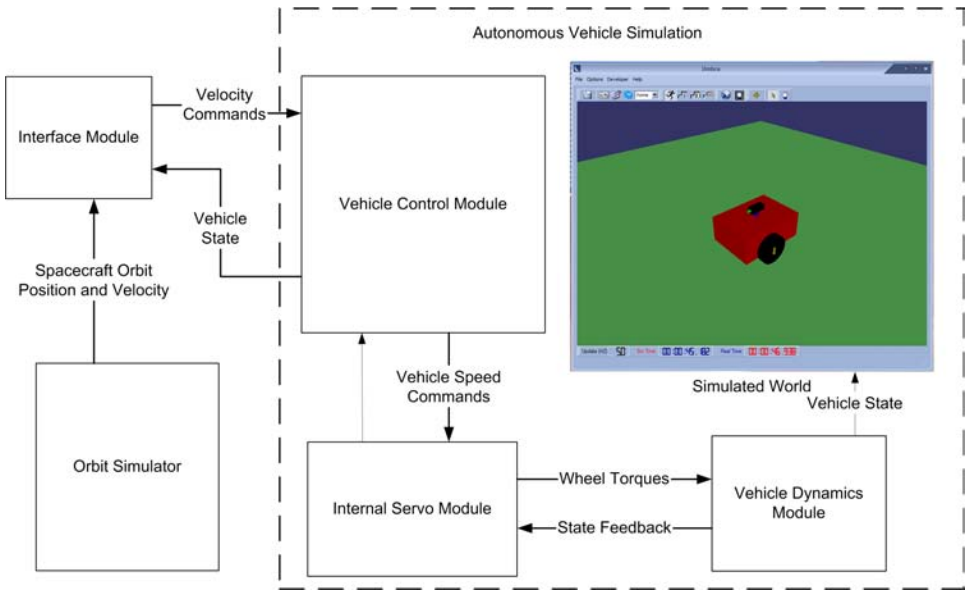


Figure 1.7: Simulated Vehicle Components Interfacing with an Orbit Simulator

UMBRA framework, yields an efficient hardware/simulation environment. This capability takes full advantage of the power of C++ in a modular form. This property is expanded to include communication over wired and wireless networks. Sandia National Labs is currently

developing such modules. However, in order to take advantage of these ideas, an accurate dynamical model of the vehicle must be created as well as the components to control it, such as an internal drive-wheel servo module. In addition the vehicle simulation must be developed using the same module that controls the real vehicle. This ensures consistency between operating with the virtual or actual vehicle. Other modules, such as an orbit simulator, are then be created to interface with either vehicle. Some of the future work that one could consider is a collaboration between the attitude simulators of the SSSL lab and the planar motion simulator of the AVS lab. This collaboration could bridge the gap for exploring coupled attitude and translational motion of spacecraft.

1.3 Outline of Thesis

The research work contained in this thesis is to begin developing the simulation capability of a two-dimensional test bed platform that will assist with researching the relative motion sensing problem. In the following chapter, the equations of motion for the ActivMedia P3-DX robotic vehicle are developed, which includes a basic wheel friction model and direct wheel torques as inputs. This provides the capability to accurately mimic the full dynamic motion of the robotic vehicle. Additionally, a wheel-speed based kinematic model is developed to provide the idealized no-slip motion of the vehicle. In the third chapter, the non-linear translational equations of motion for a spacecraft subject to orbital perturbations are developed. The equations of motion apply to a single spacecraft or to multiple space-

craft in a formation. The orbit simulation also includes a relative motion calculation and orbit perturbations such as atmospheric drag, gravitational harmonics, and solar radiation pressure. The fourth chapter illustrates how the orbit and vehicle simulations are implemented using the UMBRA framework. The key ideas explored in the fourth chapter are implementing the vehicle simulation using the same interface code that drives the actual vehicle, and also developing the internal servo module that takes vehicle speed commands as inputs, and translates them to wheel torques. This chapter also discusses the creation of a interface module between the vehicle simulation and the relative orbit module, which enables the vehicle to track spacecraft relative orbits. This interface module allows either the simulated vehicle or the actual vehicle to behave like a spacecraft in formation.

Chapter 2

Robotic Vehicle Dynamics

In order to develop a successful hardware/simulation test bed, the simulation component must accurately describe the motion of the vehicle in question. In this case, the goal for the AVS Lab is to develop a software vehicle simulation that mimics the motion of the P3-DX robotic vehicle pictured in Figure 2.1. This is achieved by developing both a kinematic speed-based model and a kinetic torque-based model. The kinematic model is a simple wheel-speed based model that is well suited to describing the ideal no-slip motion of the vehicle. The kinetic model is much more complex and includes such things as surface friction forces, normal forces, and internal forces and torques. This model is designed with wheel torques as the control input and includes a basic friction model. This adds a level of accuracy to the kinetic-based model, allowing the virtual vehicle to explore wheel slippage on different surfaces. In the kinetic model there are two different approaches to developing the vehicle equations of motion. The vehicle has two large drive wheels and one rear support wheel.

The two larger wheels are independently controlled by two motors. The third wheel is a small free-wheel or caster that only provides stability. The first approach to developing the equations of motion is to describe the vehicle as whole with two rotating wheels attached to it. This approach quickly yields the equations of motion of the vehicle. However, the wheels are examined separately to describe the motion of the vehicle as a function of the internal motor control torques. The second approach is to examine the vehicle body and two wheels as three separate bodies. These bodies are linked through the internal torques and forces that act between them. This approach is taken in this chapter, with the goal of not only obtaining the equations of motion but also describing the surface friction forces, normal forces, and internal forces as functions of the state of the vehicle. This also allows for a real-time friction model to be developed later in this chapter. Next, the kinematic model for the vehicle is described.

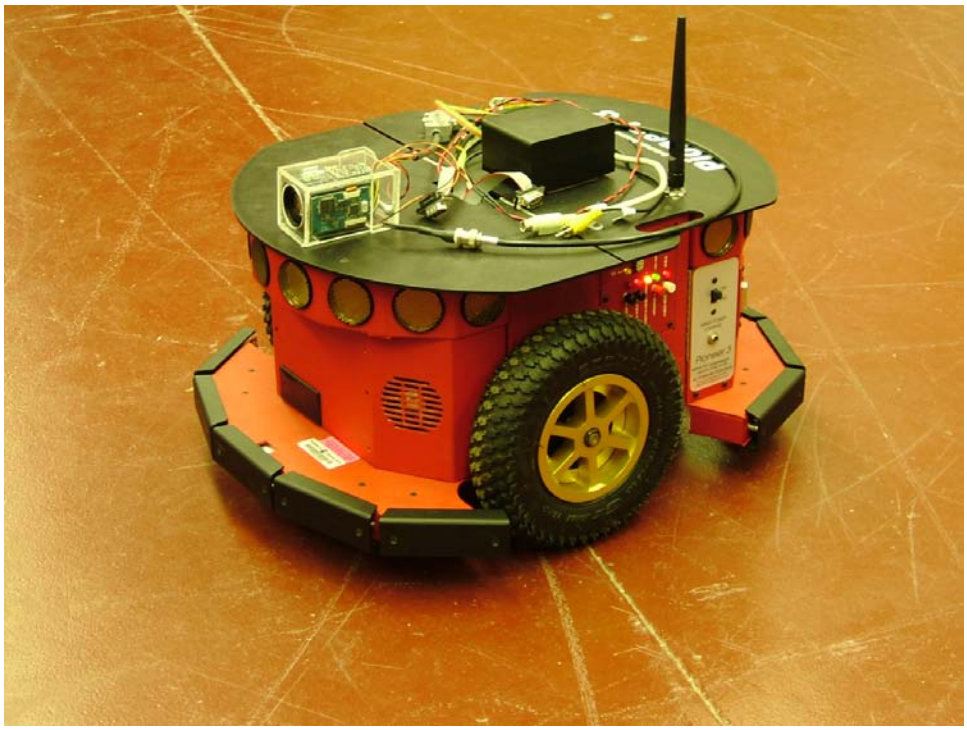


Figure 2.1: ActiveMedia Robotic Vehicle Pioneer

2.1 Kinematic Vehicle Model

Before deriving the full dynamic equations of motion we examine the kinematic equations of motion of the vehicle. This model is intended as a first iteration into simulating the ideal motion of the robotic vehicle. It is also useful in controlling the virtual vehicle using wheel speeds, where motor torques are not needed. One important item to consider in developing the kinematic model is the basic configuration and dimensions of the vehicle. A schematic diagram is included in Figure 2.2, which illustrates the drive wheel and free wheel offsets through distances $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$. These are the distances of the free wheel and driving wheel axis from the center of mass of the vehicle respectively. Here $\boldsymbol{d}_1 = -d_1\hat{\boldsymbol{b}}_2$ and $\boldsymbol{d}_2 = d_2\hat{\boldsymbol{b}}_2$ are

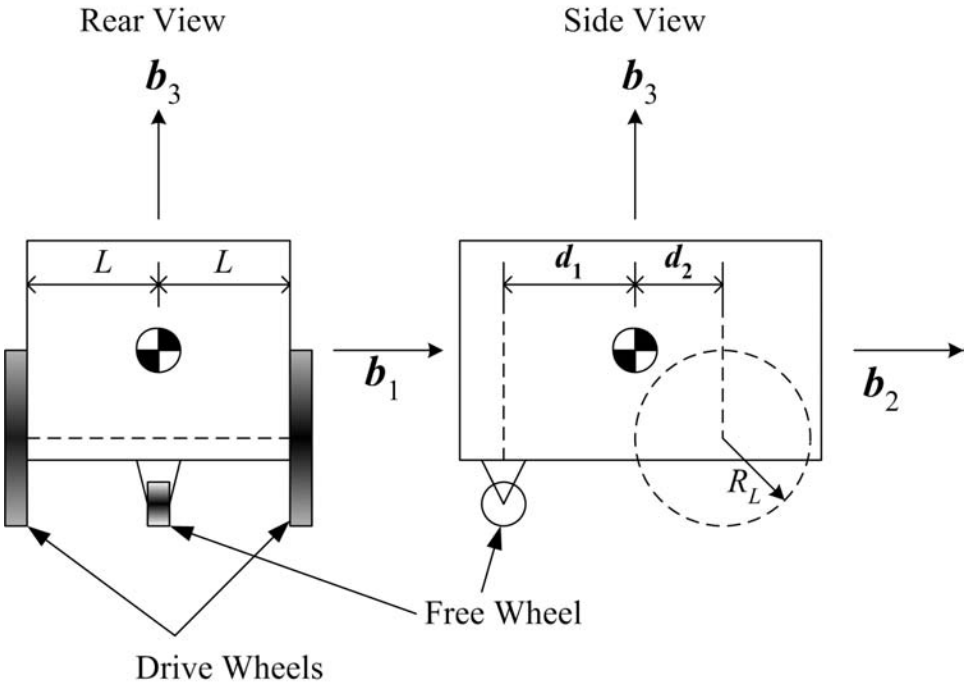


Figure 2.2: Wheel Distances on the Robotic Vehicle

distances along the $\hat{\boldsymbol{b}}_2$ axis which is the forward direction of the robotic vehicle. Also note the distance L is half the wheel-base and R_L is the left wheel radius. Additionally, Figure 2.3 defines the position and heading coordinates of the vehicle. Two key assumptions are that the vehicle body is symmetric with respect to the $\hat{\boldsymbol{b}}_2$-$\hat{\boldsymbol{b}}_3$ plane and the wheels are identical homogeneous cylinders. These assumptions and parameters are also carried through to the kinetic model. In the kinematics model the rate of change of the position and heading of the vehicle are represented in the following matrix equation.[21]

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -R_r \sin\theta & -R_l \sin\theta \\ R_r \cos\theta & R_l \cos\theta \\ \frac{R_r}{L} & -\frac{R_l}{L} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \tag{2.1}$$

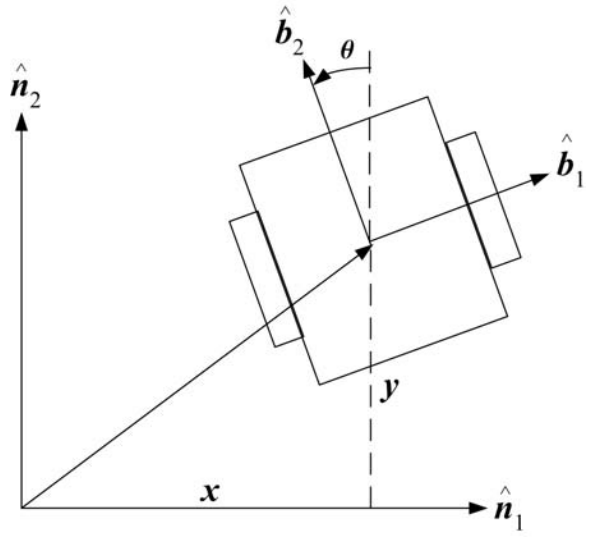


Figure 2.3: Position and Heading Coordinate Definition

Here ω_r and ω_l are the right and left wheel speeds and θ is the cart heading angle as defined in Figure 2.3. These initial equations of motion give insight into the motion of the vehicle and how movement is controlled using wheel speeds.

2.2 Kinetic Vehicle Model

To develop the dynamic equations of motion, the robotic vehicle is split into three separate bodies: two identical wheels and a vehicle body. This allows for a larger and complex problem to be broken down into three simpler problems. This approach examines the governing equations of the wheel and the vehicle body separately, and then combines them to solve for the unknown forces and the final form of the equations of motion.

2.2.1 Vehicle Reference Frames

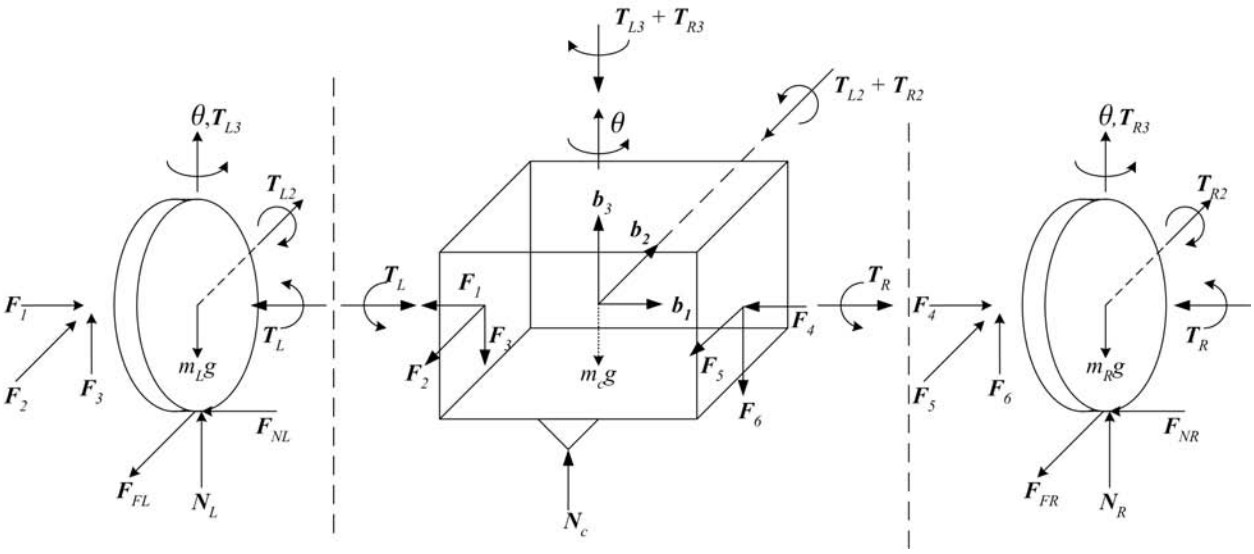


Figure 2.4: Diagram of the Robotic Vehicle

In Figure 2.4 the robotic vehicle components are displayed with the various torques and forces being applied. The torques $\boldsymbol{T}_L$ and $\boldsymbol{T}_R$ are the drive motor control torques acting on the wheels and are treated as inputs to the equations of motion. The rest of the forces and torques are unknowns and must be solved for in terms of the state of the vehicle. The principal body frame, $\mathcal{B} : \{\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\}$, is fixed to the vehicle with its origin at the center of mass of the vehicle body. Also note here that the vehicle is restricted to planar motion. Any motion in the $\hat{\boldsymbol{b}}_3$ direction is restricted to zero as well as any rotations about $\hat{\boldsymbol{b}}_1$ and $\hat{\boldsymbol{b}}_2$ axes. Another frame that is not described in the figure is the inertial frame, $\mathcal{N} : \{\hat{\boldsymbol{n}}_1, \hat{\boldsymbol{n}}_2, \hat{\boldsymbol{n}}_3\}$, whose origin is initially at the vehicle's center of mass and $\hat{\boldsymbol{n}}_i = \hat{\boldsymbol{b}}_i(t_0)$. Therefore, the

angular rate between the body frame and the inertial frame is written as follows:

$$\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} = \dot{\theta}\hat{\boldsymbol{b}}_3 \tag{2.2}$$

Also note that the free wheel is modeled here as a support force $\boldsymbol{N}_C$. Here, it is required that $\boldsymbol{N}_C$ be positive to avoid tipping the vehicle.

2.2.2 EOM of the Drive Wheel

The governing equations for the wheel are obtained by applying Euler's equation and Newtons 2nd Law. The angular momentum and torque are taken about the center of mass of the wheel and described in the body frame. This body frame $\mathcal{B}$ is also aligned with the principal inertia frame of the wheel. The angular momentum of the wheel is:[22]

$$\boldsymbol{H}_i = [I_w](\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} - \omega_i\hat{\boldsymbol{b}}_1) = \overset{\mathcal{B}}{\begin{pmatrix} I_{w1} & 0 & 0 \\ 0 & I_{w2} & 0 \\ 0 & 0 & I_{w3} \end{pmatrix}} \overset{\mathcal{B}}{\begin{pmatrix} -\omega_i \\ 0 \\ \dot{\theta} \end{pmatrix}} = I_{w3}\dot{\theta}\hat{\boldsymbol{b}}_3 - I_{w1}\omega_i\hat{\boldsymbol{b}}_1 \tag{2.3}$$

Here, the subscript i is either L or R to denote the left or right wheel respectively. Additionally, $[I_w]$ is the inertia matrix of the drive wheel. The positive wheel speeds, ω_i, are in the negative $\hat{\boldsymbol{b}}_1$ direction, which is the same direction as the control torques $\boldsymbol{T}_i$. Using the transport theorem[22] we find the rate of change of angular momentum and this is equal to

the external torques applied to the body:

$$\dot{\boldsymbol{H}}_i = \overset{\mathcal{B}}{\begin{pmatrix} -I_{w1}\dot{\omega}_i \\ -I_{w1}\dot{\theta}\omega_i \\ I_{w3}\ddot{\theta} \end{pmatrix}} = \boldsymbol{L}_i = \overset{\mathcal{B}}{\begin{pmatrix} -T_i - R_i F_{Fi} \\ T_{i2} + R_i F_{Ni} \\ T_{i3} \end{pmatrix}} \tag{2.4}$$

Newton's 2nd Law is then applied to the wheel, but in the inertial frame:

$$m_w \boldsymbol{a} = \overset{\mathcal{N}}{\begin{pmatrix} m_w \ddot{x}_L \\ m_w \ddot{y}_L \\ 0 \end{pmatrix}} = \boldsymbol{F} = \overset{\mathcal{N}}{\begin{pmatrix} (F_1 - F_{NL})\cos\theta + (F_{FL} - F_2)\sin\theta \\ (F_2 - F_{FL})\cos\theta + (F_1 - F_{NL})\sin\theta \\ N_L - m_w g + F_3 \end{pmatrix}} \tag{2.5}$$

An equivalent force equation is written for the right wheel. Note that x_i and y_i represents the position of the center of mass of the drive wheel in the inertial frame. This essentially completes the set of six equations needed to fully describe the six degrees of freedom of the wheel. However, note that the wheel has been restricted to planar motion by setting the z acceleration to zero as well as any rotation about the $\hat{\boldsymbol{b}}_2$ axis. Accompanying these equations are the no-slip conditions, which must be applied if the vehicle is to move along a surface.

$$\dot{y}_i \cos\theta - \dot{x}_i \sin\theta = R_i \omega_i \tag{2.6}$$

$$\dot{y}_i \sin\theta = \dot{x}_i \cos\theta = 0 \tag{2.7}$$

These equations are valid for both wheels and are differentiated once to describe accelerations.

2.2.3 EOM of the Vehicle Body

Euler's equation and Newtons 2nd law are applied to the cart body in similar fashion. The torques and angular momentum are taken with respect to the center of mass of the vehicle body and described in the body frame:

$$\boldsymbol{H} = [I]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} = \overset{\mathcal{B}}{\begin{pmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{pmatrix}} \overset{\mathcal{B}}{\begin{pmatrix} 0 \\ 0 \\ \dot{\theta} \end{pmatrix}} = I_3\dot{\theta}\hat{\boldsymbol{b}}_3 \tag{2.8}$$

Here $[I]$ is the inertia matrix of the vehicle body and the inertial rate of change of angular momentum in relation to the external torques is:

$$\dot{\boldsymbol{H}} = \overset{\mathcal{B}}{\begin{pmatrix} 0 \\ 0 \\ I_3\ddot{\theta} \end{pmatrix}} = \boldsymbol{L} = \overset{\mathcal{B}}{\begin{pmatrix} -N_C d_1 - T_R - T_L - (F_3 + F_6)d_2 \\ (F_6 - F_3)L - T_{R2} - T_{L2} \\ (F_1 + F_4)d_2 + (F_2 - F_5)L - T_{L3} - T_{R3} \end{pmatrix}} \tag{2.9}$$

Newton's 2nd Law is applied to the vehicle body in the inertial frame to obtain:

$$m_c\boldsymbol{a} = \overset{\mathcal{N}}{\begin{pmatrix} m_c\ddot{x} \\ m_c\ddot{y} \\ 0 \end{pmatrix}} = \boldsymbol{F} = \overset{\mathcal{N}}{\begin{pmatrix} -(F_1 + F_4)\cos\theta + (F_5 + F_2)\sin\theta \\ -(F_2 - F_5)\cos\theta - (F_1 + F_4)\sin\theta \\ N_C - m_c g - F_3 - F_6 \end{pmatrix}} \tag{2.10}$$

Combine these equations with the six equations for each wheel, yields a total of 18 equations and 17 unknown forces. Additionally, there are 5 desired unknown accelerations ($\ddot{x}$,$\ddot{y}$,$\ddot{\theta}$,$\dot{\omega}_R$,$\dot{\omega}_L$) and 4 unknown wheel accelerations ($\ddot{x}_L$,$\ddot{y}_L$,$\ddot{x}_R$,$\ddot{y}_R$). This leaves 8 unknowns, which are reduced to 4 by using the kinematic relations between the wheel positions, x_i and

y_i, and the vehicle body position, x and y. Lastly, there are 2 no-slip conditions for each wheel and a lateral no-slip condition that leaves one unknown that cannot be solved for. This unknown force is along the $\hat{\boldsymbol{b}}_1$ direction where the lateral friction forces (F_{Ni}) cannot be determined uniquely.

2.2.4 Solving for the System EOM

The strategy for solving for these forces starts with the wheel equations. From the first two components of the matrix equation 2.4, the friction forces are determined for each wheel using:

$$F_{Fi} = (-T_i + I_{w1}\dot{\omega}_i)/R_i \tag{2.11}$$

$$F_{Ni} = (-T_{i2} - I_{w1}\omega_i\dot{\theta})/R_i \tag{2.12}$$

These friction forces are used to reduce the unknowns in the first two components of equation 2.5 by two for each wheel:

$$m_w\ddot{x}_R - \frac{\dot{\omega}_R I_{w1}\sin\theta}{R_R} = \frac{(R_R F_4 + T_{R2} + I_{w1}\omega_R\dot{\theta})\cos\theta - (R_R F_5 + T_R)\sin\theta}{R_R} \tag{2.13}$$

$$m_w\ddot{y}_R - \frac{\dot{\omega}_R I_{w1}\cos\theta}{R_R} = \frac{(R_R F_4 + T_{R2} + I_{w1}\omega_R\dot{\theta})\sin\theta + (R_R F_5 + T_R)\cos\theta}{R_R} \tag{2.14}$$

$$m_w\ddot{x}_L - \frac{\dot{\omega}_L I_{w1}\sin\theta}{R_L} = \frac{(R_L F_1 + T_{L2} + I_{w1}\omega_L\dot{\theta})\cos\theta - (R_L F_2 + T_L)\sin\theta}{R_L} \tag{2.15}$$

$$m_w\ddot{y}_L - \frac{\dot{\omega}_L I_{w1}\cos\theta}{R_L} = \frac{(R_L F_1 + T_{L2} + I_{w1}\omega_L\dot{\theta})\sin\theta + (R_L F_2 + T_L)\cos\theta}{R_L} \tag{2.16}$$

Using these four equations, the four unknown pin forces (F_1,F_2,F_4,F_5) are solved for in terms of the remaining two unknown torques, T_{R2} and T_{L2}, and the known wheel motor torques:

$$F_1 = \frac{-T_{L2} - I_{w1}\omega_L\dot{\theta}}{R_L} + m_w(\cos\theta\ddot{x}_L + \sin\theta\ddot{y}_L) \tag{2.17}$$

$$F_4 = \frac{-T_{R2} - I_{w1}\omega_R\dot{\theta}}{R_R} + m_w(\cos\theta\ddot{x}_R + \sin\theta\ddot{y}_R) \tag{2.18}$$

$$F_2 = \frac{I_{w1}\dot{\omega}_L - T_L}{R_L} + m_w(-\sin\theta\ddot{x}_L + \cos\theta\ddot{y}_L) \tag{2.19}$$

$$F_5 = \frac{I_{w1}\dot{\omega}_R - T_R}{R_R} + m_w(-\sin\theta\ddot{x}_R + \cos\theta\ddot{y}_R) \tag{2.20}$$

The pin forces described here are directly substituted into the first two components of equation 2.10 and the last component of equation 2.9. The remaining components of those equations are used to solve for the normal forces. These are later used in the friction model. Before we solve for the cart accelerations, some kinematic conditions must be applied that relate the translational wheel accelerations to the accelerations for the vehicle body. These conditions essentially "attach" the wheels to the vehicle body and are written as follows.

$$y_L = y - L\sin\theta + d_2\cos\theta \tag{2.21}$$

$$y_R = y + L\sin\theta + d_2\cos\theta \tag{2.22}$$

$$x_L = x - L\cos\theta - d_2\sin\theta \tag{2.23}$$

$$x_R = x + L\cos\theta - d_2\cos\theta \tag{2.24}$$

These equations are differentiated twice to relate velocities and accelerations as needed. Using these kinematic conditions as well as the the pin forces and the no-slip conditions, the vehicle body equations 2.10 and 2.9 reduce to three equations in terms of 5 accelerations

($\ddot{x}$,$\ddot{y}$,$\ddot{\theta}$,$\dot{\omega}_R$,$\dot{\omega}_L$) and a single unknown torque sum ($T_{R2} + T_{L2}$):

$$-2\ddot{x}m_w d_2 \cos\theta - 2\ddot{y}m_w d_2 \sin\theta + (2m_w(L^2 + d_2^2) + 2I_{w3} + I_3)\ddot{\theta} - \dot{\omega}_L\left(\frac{I_{w1}L}{R}\right) + \dot{\omega}_R\left(\frac{I_{w1}L}{R}\right)$$
$$= \frac{L}{R}(T_R - T_L) - \frac{d_2}{R}(T_{L2} + T_{R2} + I_{w1}(\omega_R + \omega_L)\dot{\theta}) \quad (2.25)$$

$$\ddot{x}(2m_w + m_c) - 2m_w d_2 \cos\theta\ddot{\theta} - \dot{\omega}_R\left(\frac{I_{w1}\sin\theta}{R}\right) - \dot{\omega}_L\left(\frac{I_{w1}\sin\theta}{R}\right)$$
$$= -\sin\theta(T_R + T_L + 2m_w d_2 R\sin\theta\dot{\theta}^2) + \cos\theta(I_{w1}(\omega_R + \omega_L)\dot{\theta} + T_{R2} + T_{L2}) \quad (2.26)$$

$$\ddot{y}(2m_w + m_c) - 2m_w d_2 \sin\theta\ddot{\theta} + \dot{\omega}_R\left(\frac{I_{w1}\cos\theta}{R}\right) + \dot{\omega}_L\left(\frac{I_{w1}\cos\theta}{R}\right)$$
$$= \cos\theta(T_R + T_L + 2m_w d_2 R\sin\theta\dot{\theta}^2) + \sin\theta(I_{w1}(\omega_R + \omega_L)\dot{\theta} + T_{R2} + T_{L2}) \quad (2.27)$$

Also note that we have assumed the wheels to be identical in size and mass. The last unknown in this equation is the torque sum ($T_{R2} + T_{L2}$). In this dynamical system these torques cannot be determined uniquely. The reason is that the vehicle is treated as a rigid body and these torques act along the same axis ($\hat{\boldsymbol{b}}_2$). Structural stiffness modeling of the vehicle and its drive wheels is necessary to determine these torques uniquely. However, the kinematic and no-slip conditions for each wheel are combined to yield the no-slip conditions in terms of the full vehicle states. Applying the full no-slip conditions to equations 2.25, 2.26, and 2.27, the unknown torque sum is obtained:

$$(T_{R2} + T_{L2}) = -\frac{2I_{w1} + R^2(2m_w + m_c)}{2}(\omega_R + \omega_L)\dot{\theta} - 2m_w R d_2\ddot{\theta} \quad (2.28)$$

This equation represents the last unknown in the vehicle equations of motion. Applying equation 2.28 results in three second-order differential equations in terms of five unknown

accelerations. To complete the set of equations the no-slip condition for the full vehicle is included as well as a rotational kinematic condition. The set of equations are now arranged in matrix form in equation 2.29.

$$\begin{bmatrix} -2m_w d_2 \cos\theta & -2m_w d_2 \sin\theta & (2(m_w L^2 + I_{w3}) + I_3) & \left(\frac{I_{w1} L}{R}\right) & -\left(\frac{I_{w1} L}{R}\right) \\ (2m_w + m_c) & 0 & 0 & -\left(\frac{I_{w1}\sin\theta}{R}\right) & -\left(\frac{I_{w1}\sin\theta}{R}\right) \\ 0 & (2m_w + m_c) & 0 & \left(\frac{I_{w1}\cos\theta}{R}\right) & \left(\frac{I_{w1}\cos\theta}{R}\right) \\ -\sin\theta & \cos\theta & 0 & -R/2 & -R/2 \\ 0 & 0 & L & -R/2 & R/2 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \\ \dot{\omega}_R \\ \dot{\omega}_L \end{bmatrix}$$

$$= \begin{bmatrix} \frac{L}{R}(T_R - T_L) + \frac{Rd_2}{2}(2m_w + m_c)(\omega_R + \omega_L)\dot{\theta} \\ -\sin\theta(T_L + T_R) - \frac{R}{2}\cos\theta(2m_w + m_c)(\omega_R + \omega_L)\dot{\theta} - 2m_w \sin\theta d_2 \dot{\theta}^2 \\ \cos\theta(T_L + T_R) - \frac{R}{2}\sin\theta(2m_w + m_c)(\omega_R + \omega_L)\dot{\theta} + 2m_w \sin\theta d_2 \dot{\theta}^2 \\ 2d_2\dot{\theta}^2 L \\ 0 \end{bmatrix} \tag{2.29}$$

Which is also represented in a simple form as:

$$[A] \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \\ \dot{\omega}_R \\ \dot{\omega}_L \end{bmatrix} = [B] \tag{2.30}$$

The five accelerations then become the inverse of a five-by-five matrix, $[A]$, multiplied by the vector, $[B]$. As with any second order system, integration is done by simply breaking it

down to a larger first-order system where the state vector $\boldsymbol{X}$ and the derivative of the state are defined as:

$$\boldsymbol{X} = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \omega_R \\ \omega_L \\ \theta_R \\ \theta_L \end{bmatrix} \quad \dot{\boldsymbol{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \\ \dot{\omega}_R \\ \dot{\omega}_L \\ \omega_R \\ \omega_L \end{bmatrix} = \begin{bmatrix} X(4) \\ X(5) \\ X(6) \\ A^{-1}B \\ X(7) \\ X(8) \end{bmatrix} \tag{2.31}$$

Note that it is desirable to keep track of wheel angles, θ_R and θ_L, for the purpose of simulating encoder outputs, as well as incorporating wheel movement in the graphical simulation. These equations represent the full dynamic no-slip motion of the robotic vehicle. However, in order to accurately mimic the motion of the vehicle on general surfaces, it is also necessary to include a basic friction model that allows the wheels to slip.

2.3 Numerical Vehicle Simulations

In this section, the goal is to numerically simulate the simple kinematics model and the torque-based kinetic model. This gives us a better understanding of both models and allows

us to verify basic movement of the vehicle. The kinematics model, which is described in equation 2.1, is essentially a wheel-speed based model, while the kinetic model takes wheel torques as inputs. For these numerical simulations it is desirable to include some realistic parameters for the robotic vehicle in question, such as mass, size, and moments of inertia. Estimates of these parameters are obtained by consulting the ActiveMedia robotics manual as well as through some hand measurements. Figure 2.5 shows the basic dimensions of the vehicle used in the simulation and Table 2.1 lists additional vehicle parameters. For calculating the moment of inertia of the wheel, it is assumed to be a homogeneous cylinder and the appropriate inertia equation is used. The vehicle body is estimated as a homogeneous rectangular body. Using these vehicle parameters, the two different vehicle models are simulated. The equations of motion for each of these models are integrated and the vehicle

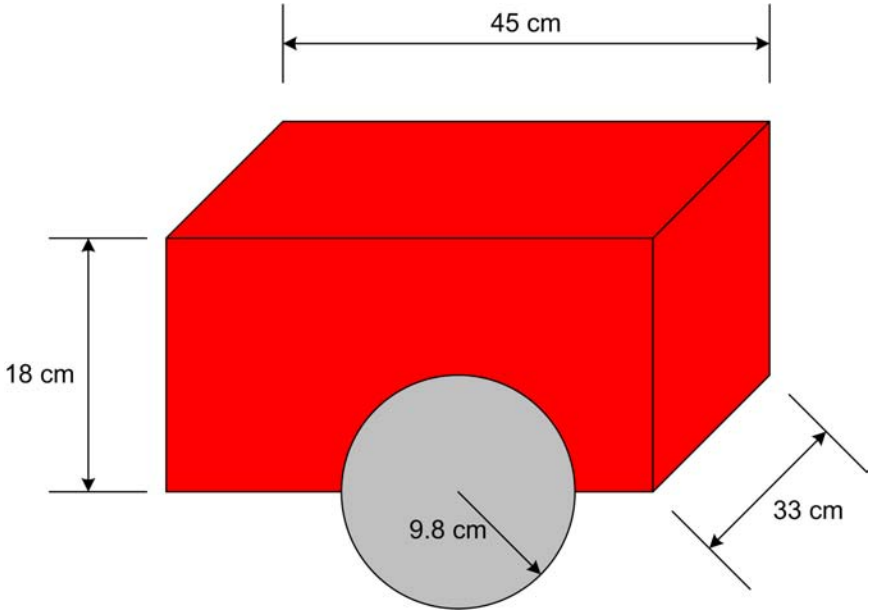
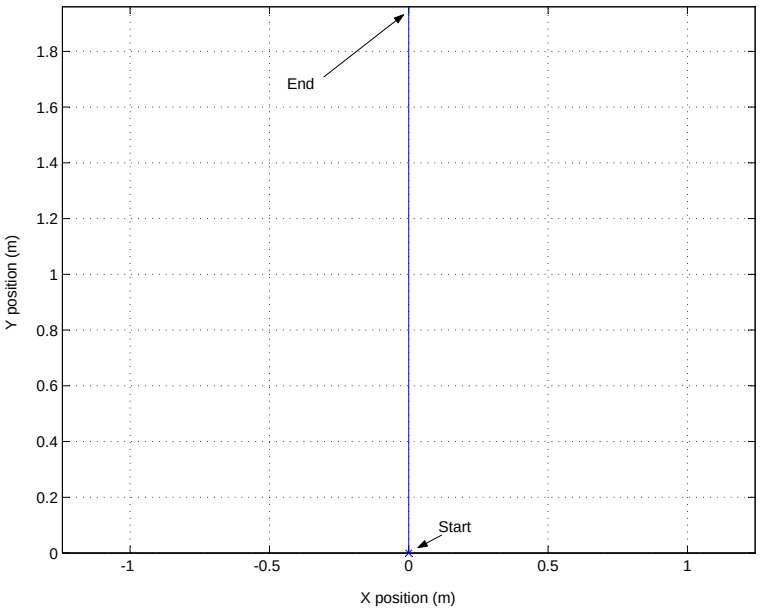


Figure 2.5: Diagram of Robotic Vehicle with approximate dimensions

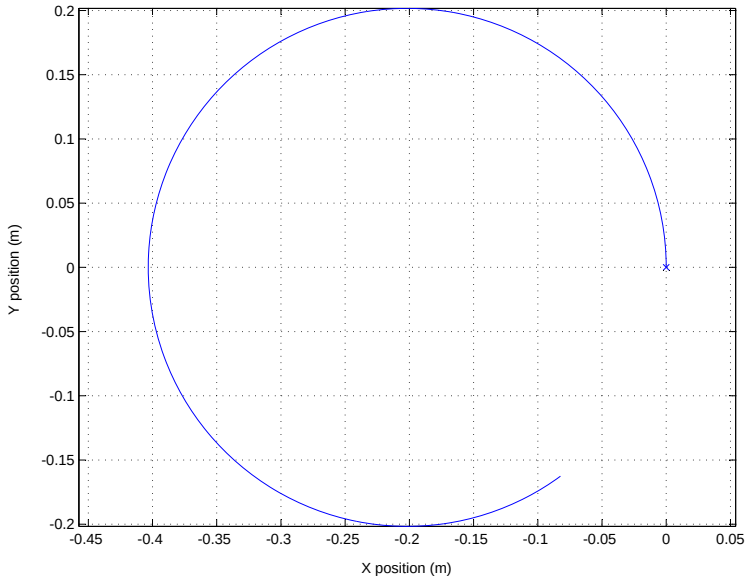
Table 2.1: Estimates of Vehicle Parameters

m_c	Mass of the vehicle body.	8 kg
m_w	Mass of the vehicle wheels.	0.5 kg
R	Radius of the wheels	0.098 m
I_{w1}	Inertia of the wheel about the spin axis ($\hat{\boldsymbol{b}}_1$).	0.0025 kgm^2
I_{w3}	Inertia of the wheel about the vertical axis ($\hat{\boldsymbol{b}}_3$).	0.001305 kgm^2
I_3	Inertia of the vehicle body about the vertical axis ($\hat{\boldsymbol{b}}_3$).	0.2 kgm^2
d_1	Distance of the third wheel from the center of mass.	0.225 m
d_2	Distance of the wheel axis from the center of mass.	0.05 m
L	Half the distance between left and right wheels.	0.165 m

states are plotted. For the kinematics model two simple cases are chosen. First, equal wheel speed commands are issued that would instinctively move the vehicle in the forward ($\hat{\boldsymbol{b}}_2$) direction. The second case features different wheel speeds that move the vehicle in a circle. The position plots for both cases are shown in Figure 2.6. Using equation 2.29, the dynamic response of the robotic vehicle to two equal constant motor torques is plotted in Figure 2.7. As expected, the vehicle moves forward in the y-direction at a quadratic rate, with the speed increasing at a linear rate. The second case in Figure 2.8 shows the vehicle accelerating on a constant radius circle with a linearly increasing heading rate. Note that in the plot for the wheel speeds, red corresponds to the right wheel and blue to the left wheel. These simple test cases help verify that our kinematic and kinetic models work as expected.

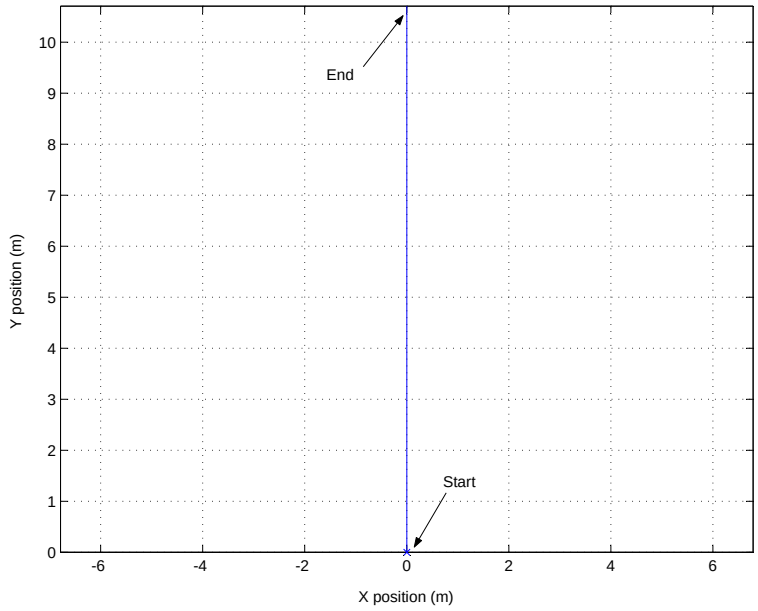


(a) Kinematic Model Response ($\omega_r = \omega_l = 1$ rad/s)

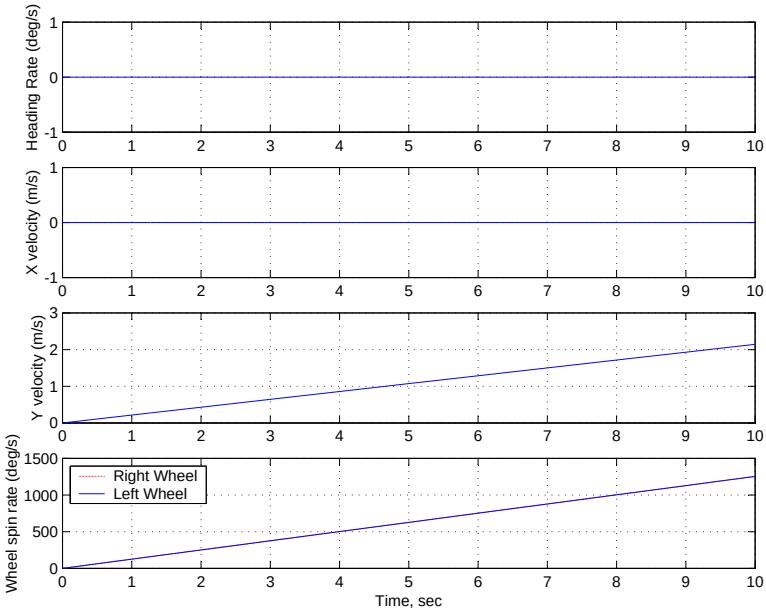


(b) Kinematic Model Response ($\omega_l = 0.1$ rad/s, $\omega_r = 1$ rad/s)

Figure 2.6: X and Y Position Plots of the Kinematic Model Response

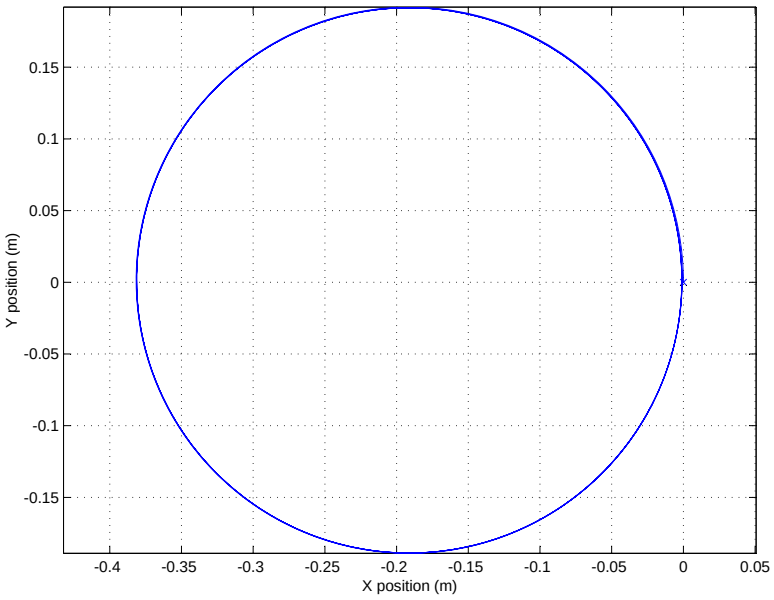


(a) X and Y Position Plots

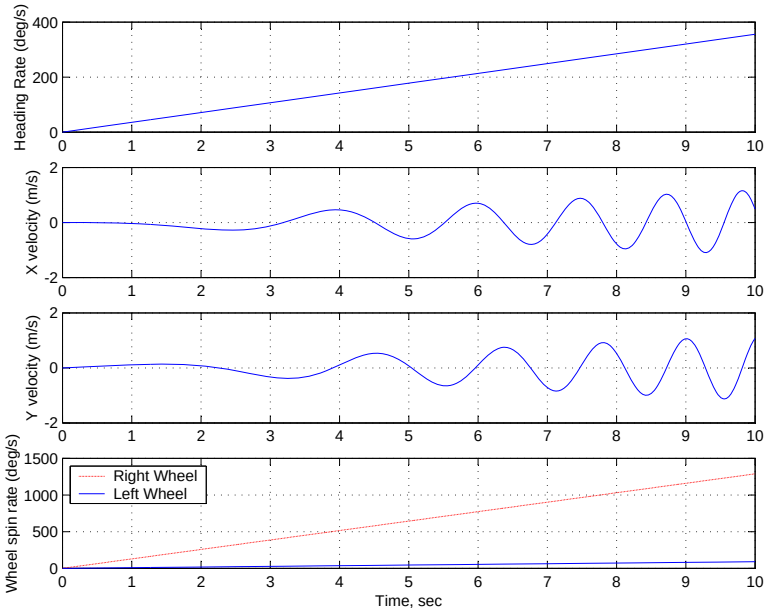


(b) Velocity and Wheel Speeds

Figure 2.7: Dynamic Model Response ($T_r = T_l$= 0.1 Nm)



(a) X and Y Position Plots



(b) Velocity and Wheel Speeds

Figure 2.8: Dynamic Model Response (T_r= 0.1 Nm, T_l= 0.01 Nm)

2.4 Friction Model

The friction model chosen for the simulation is a basic static friction model. Later versions or iterations of the simulation will possibly include a more sophisticated friction model. The goal of this friction model is to add a layer of accuracy to the kinetic vehicle model. Specifically, the friction model allows the drive wheels to slip when the vehicle is moving on a general surface. This adds realistic error to the kinetic vehicle simulation. The static friction model is based on the standard physics relation.

$$F_f \leq \mu N \tag{2.32}$$

Here F_f is the maximum possible surface friction force in relation to the normal force acting at that point. If the friction force is found to exceed this amount then there would be slippage. In this case the friction force would be replaced by the maximum friction force, μN, and the accelerations acting on the vehicle are recomputed. Before this is accomplished the friction force equations are needed. Equation 2.11 describes these friction forces for each wheel in terms of various accelerations and states. However, as noted before, the torques T_{R2} and T_{L2} cannot be solved for explicitly. Therefore, the lateral friction forces F_{NR} and F_{NL} cannot be described explicitly, only their sum. Combined with equation 2.28, the friction forces in their final form are obtained.

$$(F_{NR} + F_{NL}) = \frac{R}{2}(2m_w + m_c)(\omega_R + \omega_L)\dot{\theta} + 2m_w d_2 \ddot{\theta} \tag{2.33}$$

$$F_{FR} = \frac{-T_R - I_{w1}\dot{\omega}_R}{R} \tag{2.34}$$

$$F_{FL} = \frac{-T_L - I_{w1}\dot{\omega}_L}{R} \tag{2.35}$$

In addition to these equations it is necessary to obtain the normal force relations for each wheel, N_R and N_L, which is achieved by taking the first two elements of equation 2.9 and the last components of equations 2.10 and 2.5. This results in five equations from which five unknown forces are solved for (N_C,F_3,F_6,N_L,N_R). Using equation 2.28 again the normal forces are obtained in their final form:

$$N_L = m_w g + \frac{1}{2L(d_1+d_2)}(gLd_1m_c - L(T_R+T_L)) + \frac{T_{R2}+T_{L2}}{2L} \tag{2.36}$$

$$N_R = m_w g + \frac{1}{2L(d_1+d_2)}(gLd_1m_c - L(T_R+T_L)) - \frac{T_{R2}-T_{L2}}{2L} \tag{2.37}$$

These equations complete the set of required force computations in the friction model. The next step is to define the logic for checking for slippage. In this model, each required friction force is compared to the corresponding maximum allowable friction friction force obtained from applying equation 2.32. In the case of lateral friction, the total lateral friction force is compared to the sum of the normal forces multiplied by a lateral friction coefficient, μ_N. Again this is assuming identical wheels. If any of the friction forces are greater then the maximum then they are replaced with the maximum friction force. Therefore the five cart accelerations must be recomputed based on these new forces. These equations defines the new motion of the cart as the wheels slip. The following equations are obtained by applying

Newtons 2nd law and Euler's equation on the vehicle as a whole:

$$\ddot{x} = \frac{1}{(2m_w + m_c)}((F_{FL} + F_{FR})\sin\theta - (F_{NL} + F_{NR})\cos\theta) \tag{2.38}$$

$$\ddot{y} = \frac{1}{(2m_w + m_c)}(-(F_{FL} + F_{FR})\cos\theta - (F_{NL} + F_{NR})\sin\theta) \tag{2.39}$$

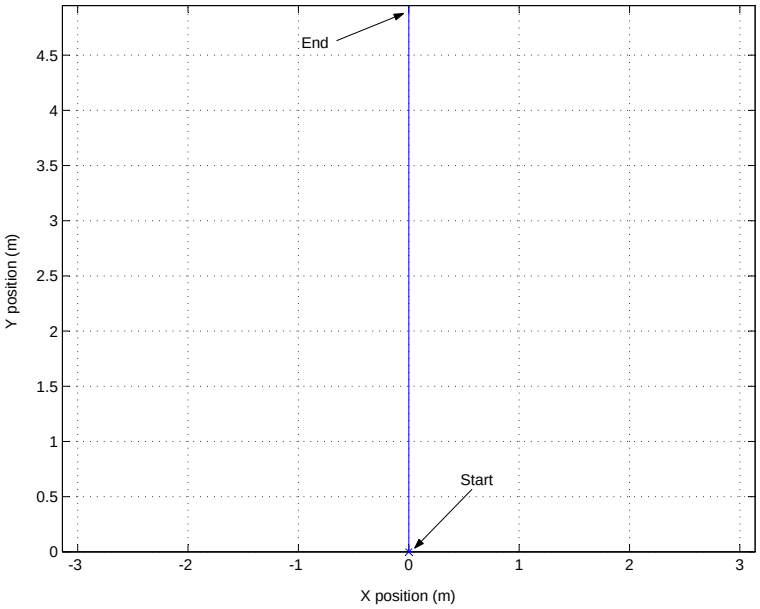
$$\ddot{\theta} = ((F_{NL} + F_{NR})d_2 + (F_{FL} + F_{FR})L)/(I_3 + 2I_{w3} + 2mL^2) \tag{2.40}$$

$$\dot{\omega}_R = (-T_R - F_{FR}R)/I_{w1} \tag{2.41}$$

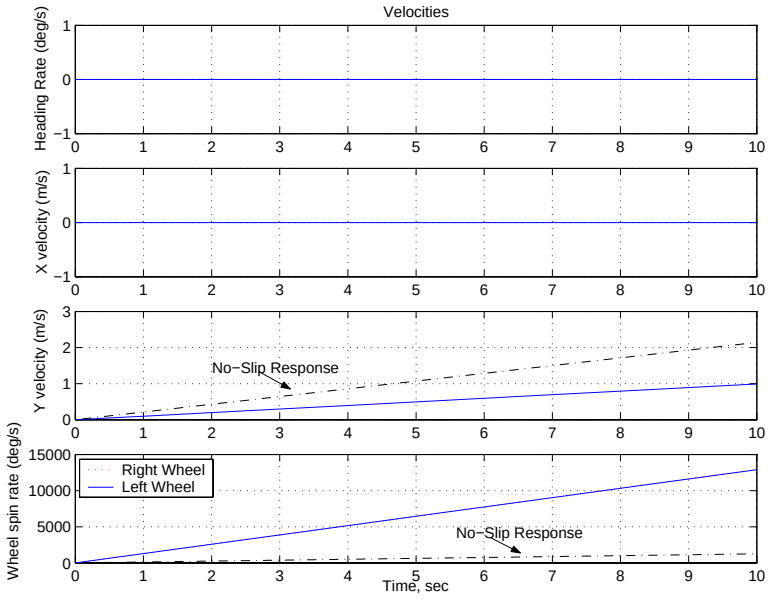
$$\dot{\omega}_L = (-T_L - F_{FL}R)/I_{w1} \tag{2.42}$$

2.5 Numerical Simulations with Vehicle Slippage

The goal of this section is to examine this friction model using simple test cases when the vehicle slips to verify the algorithm and code. In the same MATLAB environment as before, the friction model is implemented into the equations of motion. The resulting friction and normal forces are plotted and examined. To compare to a "no-slip" situation, the same two test cases that were previously used in Figures 2.7 and 2.8 are considered. These two test cases also allow us to explore the forward slipping and the lateral slipping independently. For the first test case, the torques remain the same, but the forward friction coefficients are lowered to a point where the vehicle slips, while ignoring the lateral friction coefficient. In the second test case, the forward friction coefficients are kept high enough not to slip, while the lateral friction coefficients are lowered to a point where the vehicle slips side to side.

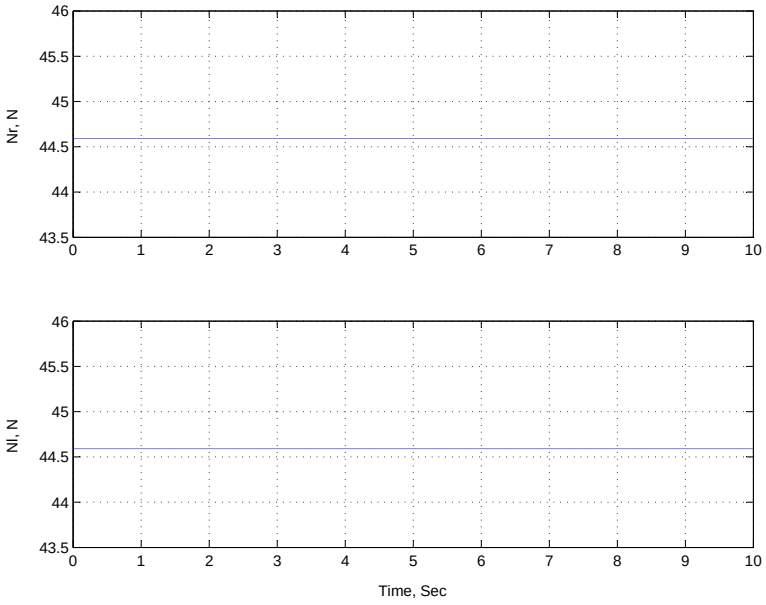


(a) Ground Track Plot

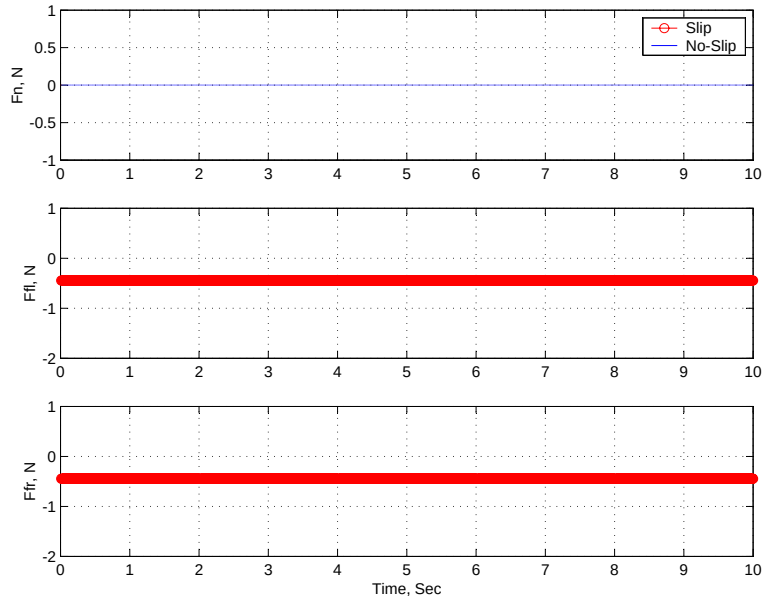


(b) Velocity and Wheel Speeds

Figure 2.9: Dynamic Model Response to Forward Slippage (μ_{fR}=μ_{fL}=0.01)

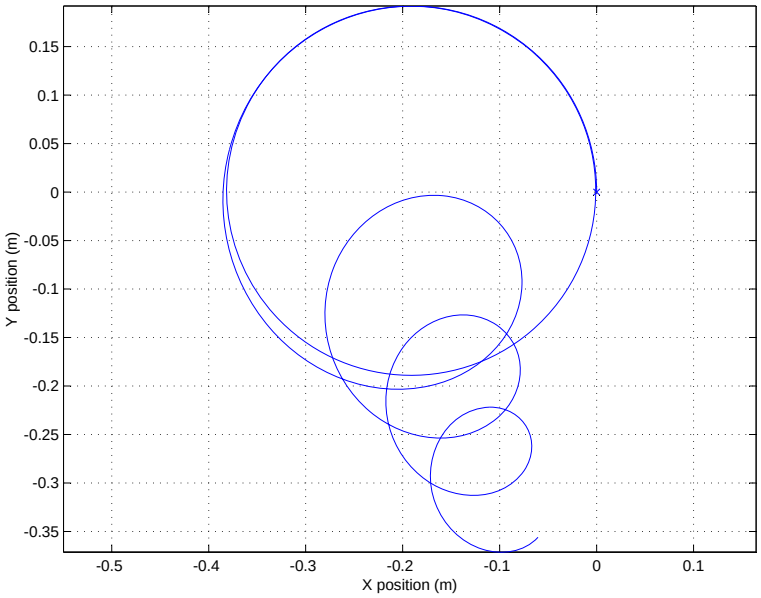


(a) Normal Forces

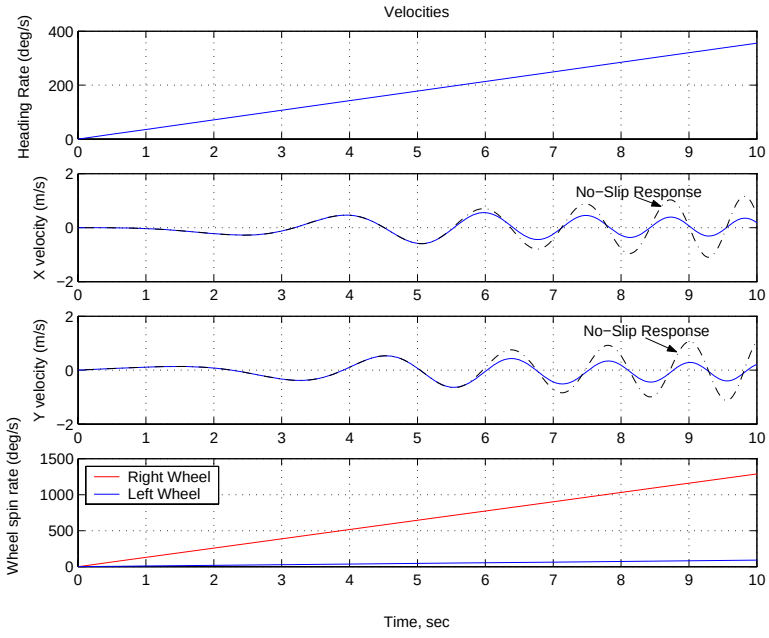


(b) Friction Forces

Figure 2.10: Friction and Normal Forces with Forward Slippage (μ_{fR}=μ_{fL}=0.01)

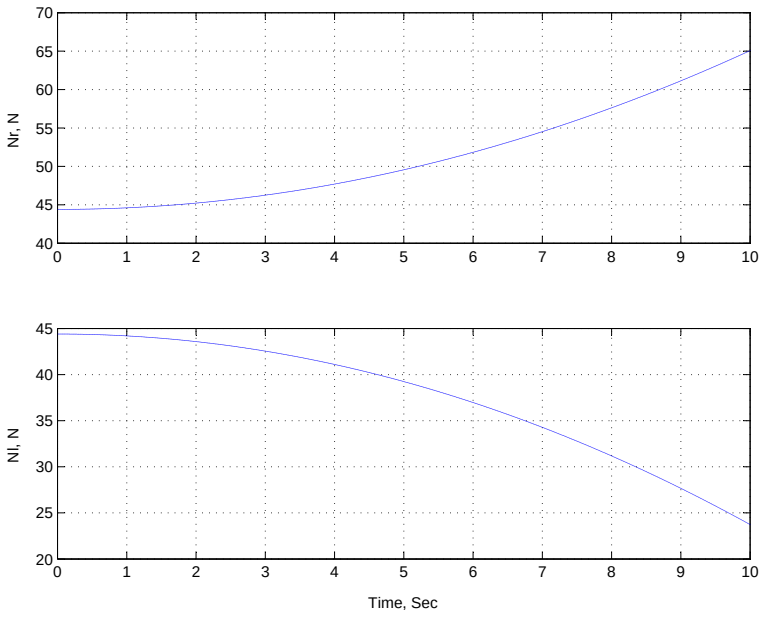


(a) Ground Track Plot

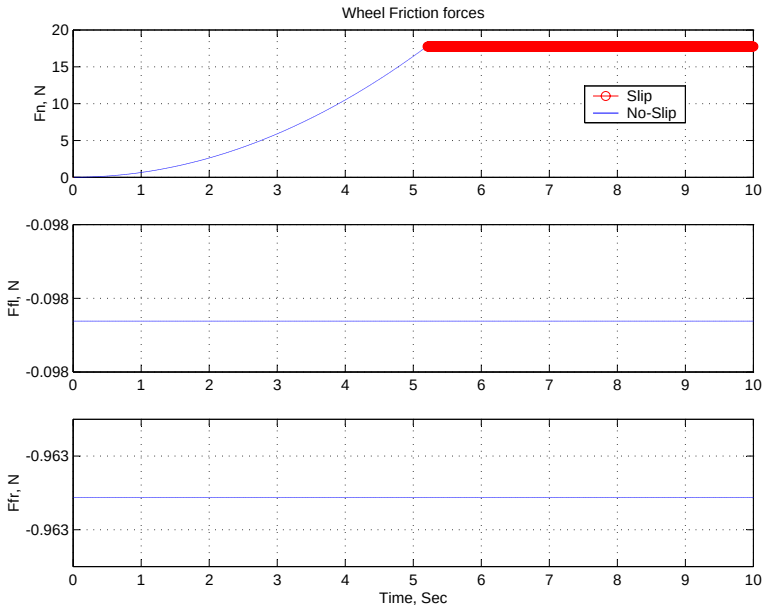


(b) Velocity and Wheel Speeds

Figure 2.11: Dynamic Model Response to Lateral Slippage (μ_N=0.2)



(a) Normal Forces



(b) Friction Forces

Figure 2.12: Friction and Normal Forces with Lateral Slippage (μ_N=0.2)

In the first test case presented in Figures 2.9 and 2.10, we observe that wheels slip immediately due to the constant torque input. Slippage is noted as a thick red line in plot 2.10(b) of the friction forces. Also, in comparing the plots to the results of the no-slip case in Figure 2.7, the wheels spin up faster and the vehicle does not move as far. This of course makes sense when a vehicle that is trying to accelerate too quickly. In the second case, the vehicle is accelerating around a circular path. The wheels slip in the lateral direction and the vehicle leaves this circular path. The results are shown in Figures 2.11 and 2.12. Another interesting note is that the normal forces on the "inside" wheel decrease and coincidently the normal force on the "outside" wheel increase. This property is synonymous with the vehicle's tendency to flip over to the side as it is accelerating through this circle. These simple test cases again provide some interesting insight into the vehicle's behavior while it is slipping. Also note that in a normal lab environment the vehicle may never slip, but this tool allows us to explore different environments that the actual vehicle is unable to reach. The equations of motion as well as this friction model are implemented in the UMBRA simulation environment to interface with other simulations.

Chapter 3

Relative Orbit Simulation

In this chapter, the relative orbit concept is explored by developing the equations of motion for multiple spacecraft in a formation. Traditionally, the relative motion of spacecraft is studied when they are in nearly circular orbits. For these cases, the Clohessy-Wiltshire-Hill equations are commonly used.[9,13,22] These equations linearize the relative motion dynamics with respect to a *constantly* rotating reference frame. They have a well-known analytical solution to the unperturbed motion which decouples the orbital in-plane and out-of-plane motions. The CW equations are convenient to develop rendezvous and near-circular formation flying control laws.[14,25] However, due to the effects of orbit perturbations on the relative orbit frame, it is necessary to avoid the linearization and constant orbit rate issues of the CW equations. Therefore, a full non-linear simulation of the spacecraft is developed with the perturbations included as inertial acceleration vectors. The relative motion of the spacecraft is then computed by processing the inertial solution. One of the objectives of this

orbit simulation is to describe the perturbation effects on the relative motion of spacecraft as accurately as possible.

3.1 Non-Linear Equations of Motion

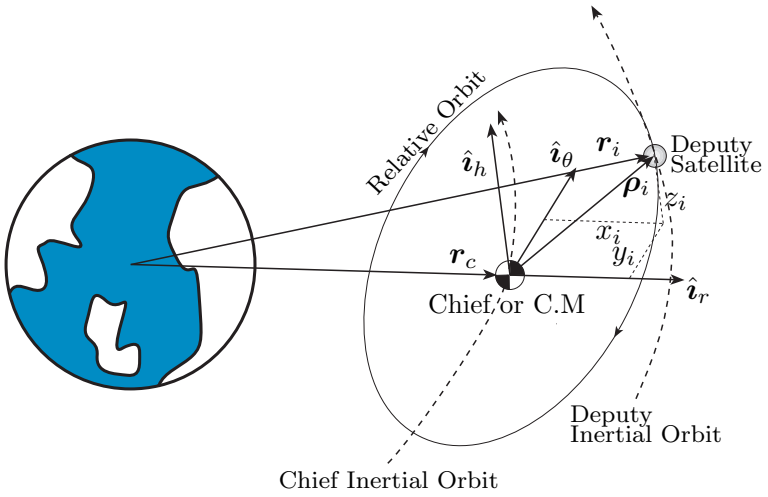


Figure 3.1: Diagram of the Relative Motion Problem

Figure 3.1 describes the set-up of the relative motion problem and the notation to be used in the rest of this chapter. The inertial equations of motion of a satellite are given by

$$\ddot{\boldsymbol{r}}_i = -\frac{\mu}{{r_i}^3}\boldsymbol{r}_i + \boldsymbol{a}_{d_i} \tag{3.1}$$

where $\boldsymbol{a_{d_i}}$ is the disturbance acceleration acting on the i^{th} satellite. Of interest is the motion of a particular satellite relative to the formation center of mass $\boldsymbol{r_c}$ or another satellite in the formation. These two types of relative motion descriptions are well suited for the formation maintenance and rendezvous and docking applications. In addition, the influence of the

disturbances on the center of mass is also of interest. Let there be N general spacecraft present with their inertial position vectors given by $\boldsymbol{r}_i$, while

$$\boldsymbol{r_c} = \frac{1}{M}\sum_{i=1}^{N} m_i \boldsymbol{r}_i \tag{3.2}$$

is the formation inertial center of mass vector. Here M is the total system mass and m_i is the mass of the i_{th} spacecraft. The velocity of the center of mass is computed by differentiating the center of mass condition once to obtain:

$$\dot{\boldsymbol{r}}_{\boldsymbol{c}} = \frac{1}{M}\sum_{i=1}^{N} m_i \dot{\boldsymbol{r}}_i \tag{3.3}$$

To describe the motion relative to another satellite, simply replace the $\boldsymbol{r}_c$ vector with the $\boldsymbol{r}_i$ vector of the satellite of interest. The Hill frame unit direction vectors $\mathcal{H} : \{\hat{\boldsymbol{\imath}}_r, \hat{\boldsymbol{\imath}}_\theta, \hat{\boldsymbol{\imath}}_h\}$ are defined through:

$$\hat{\boldsymbol{\imath}}_r = \frac{\boldsymbol{r}_c}{r_c} \qquad \hat{\boldsymbol{\imath}}_\theta = \hat{\boldsymbol{\imath}}_h \times \hat{\boldsymbol{\imath}}_r \qquad \hat{\boldsymbol{\imath}}_h = \frac{\boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c}{|\boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c|}$$

The Hill frame is aligned with the radial, out-of-plane, and along-track directions of the chief orbit. In the CW equations, this frame is treated as having a constant rotation rate. However, when orbital perturbations are included the actual $\dot{\boldsymbol{r}}_c$ is taken into account. The rotation matrix $[HN]$, which rotates vector components taken with respect to the inertial frame to vector components taken with respect to the the Hill frame, is defined as

$$[HN] = \begin{bmatrix} \hat{\boldsymbol{\imath}}_r & \hat{\boldsymbol{\imath}}_\theta & \hat{\boldsymbol{\imath}}_h \end{bmatrix}^T \tag{3.4}$$

Defining the relative position vector as $\boldsymbol{\rho}_i = \boldsymbol{r}_i - \boldsymbol{r}_c$, this vector is mapped between inertial and Hill frame vector components using ${}^{\mathcal{H}}\boldsymbol{\rho}_i = [HN]\,{}^{\mathcal{N}}\boldsymbol{\rho}_i$.

3.2 Computing Relative Motion

The purpose of this orbit simulation is to describe the relative motion of satellites using the non-linear *inertial* equations of motion. Therefore, the first step is to obtain the inertial position and velocity vectors of each of the spacecraft from the initial relative vectors $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_i'$. To achieve this, the rotation matrix $[NH]$ is calculated based on the initial chief orbit of the formation using equation 3.4. These initial chief or CM conditions are supplied by the user. In addition, the user supplies the initial desired Hill-frame coordinates of each of the spacecraft. The relative position and velocity vectors are written as follows.[22]

$$ {}^{\mathcal{H}}\boldsymbol{\rho}_i = \begin{bmatrix} x & y & z \end{bmatrix}^T \qquad {}^{\mathcal{H}}\boldsymbol{\rho}_i' = {}^{\mathcal{H}}\frac{\mathrm{d}}{\mathrm{d}t}(\boldsymbol{\rho}_i) = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \tag{3.5} $$

Using this information, the position of each spacecraft in the ECI frame is written as[22]

$$ {}^{\mathcal{N}}\boldsymbol{r}_{di} = [NH] \begin{bmatrix} r_c + x \\ y \\ z \end{bmatrix} \tag{3.6} $$

Here r_c is the chief orbit radius. The velocity vector is:[22]

$$ {}^{\mathcal{N}}\dot{\boldsymbol{r}}_{di} = {}^{\mathcal{N}}\dot{\boldsymbol{r}}_c + [NH] \left(\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \boldsymbol{\omega}_{\mathcal{H}/\mathcal{N}} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right) \tag{3.7} $$

Here $\boldsymbol{\omega}_{\mathcal{H}/\mathcal{N}} = \boldsymbol{\omega}$ is the angular velocity of the Hill Frame with respect to the inertial frame. This is defined initially as ${}^{\mathcal{H}}\boldsymbol{\omega} = \begin{bmatrix} 0 & 0 & \dot{f} \end{bmatrix}^T$, where $\dot{f}$ is the true anomaly rate of the chief

orbit and is defined as:

$$\dot{f} = \frac{|\boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c|}{r_c^2} = \frac{h}{r_c^2} \tag{3.8}$$

This relationship comes from the definition of angular momentum of a general orbit.[22] Note that these equations are only implemented at the beginning of the simulation to set up the initial inertial vectors properly from the relative orbit inputs. During the integration process of the inertial equations of motion, it is desirable to obtain the relative motion of a particular spacecraft with respect to the center of mass of the formation or another spacecraft. This is achieved by applying the reverse process to that described above to obtain the relative position and velocity vectors. The inertial relative position is defined as:

$$^{\mathcal{N}}\boldsymbol{\rho}_i = {}^{\mathcal{N}}\boldsymbol{r}_{di} - {}^{\mathcal{N}}\boldsymbol{r}_c \tag{3.9}$$

and is mapped into the Hill frame using:

$$^{\mathcal{H}}\boldsymbol{\rho}_i = [HN]\,{}^{\mathcal{N}}\boldsymbol{\rho}_i \tag{3.10}$$

For the relative velocity a similar process is done:

$$^{\mathcal{H}}\boldsymbol{\rho}_i' = [HN]\,{}^{\mathcal{N}}\dot{\boldsymbol{\rho}}_i - \boldsymbol{\omega} \times {}^{\mathcal{H}}\boldsymbol{\rho}_i \tag{3.11}$$

However, since we are dealing with perturbations that cause the frame to rotate about the along-track and radial directions, it is desirable to compute the angular velocity components more rigorously. The second and third components of $\boldsymbol{\omega}$ are obtained through applying the transport theorem to position vector of the chief spacecraft, $\boldsymbol{r}_c$:

$$\dot{\boldsymbol{r}}_c = \frac{^{\mathcal{H}}\mathrm{d}}{\mathrm{d}t}(\boldsymbol{r}_c) + \boldsymbol{\omega} \times \boldsymbol{r}_c = \dot{r}_c\hat{\boldsymbol{\imath}}_r - r_c\omega_2\hat{\boldsymbol{\imath}}_h + r_c\omega_3\hat{\boldsymbol{\imath}}_\theta \tag{3.12}$$

Now ω_2 and ω_3 are obtained as follows:

$$\omega_2 = \frac{\dot{\boldsymbol{r}}_c}{r_c} \cdot \hat{\boldsymbol{\imath}}_\theta \qquad\qquad \omega_3 = -\frac{\dot{\boldsymbol{r}}_c}{r_c} \cdot \hat{\boldsymbol{\imath}}_h \tag{3.13}$$

The final component of the angular velocity, ω_1, is obtained by taking the inertial derivative of the unit direction vector $\hat{\boldsymbol{\imath}}_h$:

$$\overset{\mathcal{N}}{\frac{\mathrm{d}}{\mathrm{d}t}}(\hat{\boldsymbol{\imath}}_h) = \omega_2 \hat{\boldsymbol{\imath}}_r - \omega_1 \hat{\boldsymbol{\imath}}_\theta = \frac{\boldsymbol{r}_c \times \ddot{\boldsymbol{r}}_c}{h_c} - \hat{\boldsymbol{\imath}}_h \frac{\dot{h}_c}{h_c} \tag{3.14}$$

And ω_1 is obtained as follows.

$$\omega_1 = -\left(\frac{\boldsymbol{r}_c \times \ddot{\boldsymbol{r}}_c}{|\boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c|} \right) \cdot \hat{\boldsymbol{\imath}}_\theta \tag{3.15}$$

Lastly, it is also desirable to compute the relative accelerations acting on each spacecraft. These acceleration components are useful for studying the relative motion dynamics of the spacecraft formation. To obtain these accelerations, we must revisit equation 3.9 which is rearranged to yield:

$$\boldsymbol{r}_i = \boldsymbol{r}_c + \boldsymbol{\rho}_i \tag{3.16}$$

This equation is differentiated twice to yield the relative acceleration of the spacecraft in the Hill frame.

$$\boldsymbol{\rho}_i'' = \ddot{\boldsymbol{r}}_i - \ddot{\boldsymbol{r}}_c - \boldsymbol{\omega} \times \boldsymbol{\rho}_i' - \dot{\boldsymbol{\omega}} \times \boldsymbol{\rho}_i - \boldsymbol{\omega} \times (\dot{\boldsymbol{r}}_i - \dot{\boldsymbol{r}}_c) \tag{3.17}$$

To obtain $\dot{\boldsymbol{\omega}}$, a simple numerical differentiation scheme is used.

$$\dot{\boldsymbol{\omega}} = \frac{\boldsymbol{\omega}_k - \boldsymbol{\omega}_{(k-1)}}{h} \tag{3.18}$$

The complete relative orbit simulator now has the capability to examine the relative motion of spacecraft and apply control solutions or orbit tracking using the robotic vehicle. Figure

3.2 shows a plot of relative orbit using the numerical simulation described above. In this plot, two spacecraft have an initial separation distance of 200 meters with a circular chief orbit that has an altitude of 300 km. For this test case, the initial Hill-frame positions and velocities are selected such that the resulting motion is bounded with a circular projection in the local horizontal plane. This case is chosen due to the increasing interest in radar interferometry applications.[8]

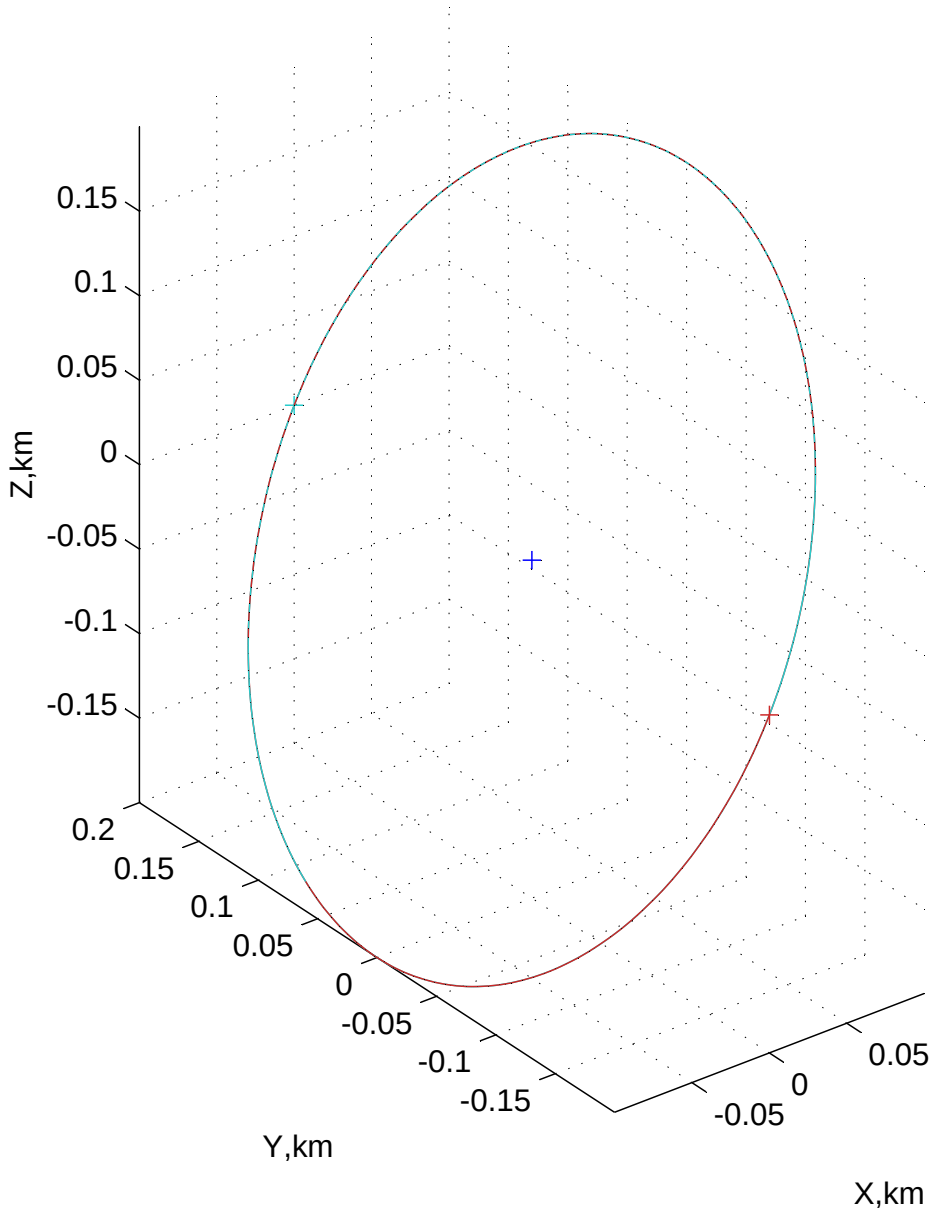


Figure 3.2: Plot of Relative Orbit Between 2 Spacecraft

3.3 Orbit Perturbations

In this section the orbital perturbations that are implemented in the relative orbit simulation are discussed. These typical orbital perturbations have a large effect on the relative motion of two closely orbiting satellites. In an effort to examine this problem in a general way, the effects of atmospheric drag, J_2 perturbations, and solar radiation drag are included as disturbance accelerations in equation 3.1. For reference, Table 3.1 describes the initial conditions and spacecraft parameters that are used in the simulations following the rest of this chapter.

Table 3.1: Initial Conditions and Spacecraft Parameters

r_c	Initial Chief Orbit Altitude	300 km
$\|\rho_i\|$	Initial Separation Distance	200 m
m	Spacecraft Mass	50 kg
C_{d1} and C_{d2}	Coefficient of Drag for each spacecraft	2.6 or 2.0
r_{sat}	Cylindrical Radius of each spacecraft	0.5 m
h	Height of each spacecraft	1.5 m
A_1 and A_2	Cross-sectional area for each spacecraft	0.7854 or 1.5000 m^2

3.3.1 Gravitational Zonal Harmonics

The J_2 through J_6 perturbations arise from the fact that the Earth is not a perfect sphere, but rather ellipsoidal in shape. Of these gravitational disturbances, the 2$^{\text{nd}}$ order zonal harmonic, called the J_2 term, is about three orders of magnitude larger than the remaining harmonics. It provides the dominant formation flying perturbation for spacecraft of equal type and build.[2] These formation flying spacecraft are typically envisioned to be flying about 1 km apart, or farther.[7] However, with this simulation much smaller separation distances can also be examined based on user input. With these smaller separation distances the differential J_2 influence becomes even smaller. An interesting problem to examine is how these differential accelerations compare to other orbital perturbations such as differential atmospheric drag and differential solar radiation pressure.

The inertial disturbance acceleration vector due to J_2 through J_6 is modeled as a function of inertial position and contains six zonal harmonic terms. However, only the first zonal harmonic has a significant contribution, which is expressed as.[22]

$$\boldsymbol{a}_{J_2} = \frac{-3}{2} J_2 \left(\frac{\mu}{r^2}\right) \left(\frac{r_{\text{eq}}}{r}\right)^2 \begin{pmatrix} \left(1 - 5\left(\frac{Z}{r}\right)^2\right) \frac{X}{r} \\ \left(1 - 5\left(\frac{Z}{r}\right)^2\right) \frac{Y}{r} \\ \left(3 - 5\left(\frac{Z}{r}\right)^2\right) \frac{Z}{r} \end{pmatrix} \tag{3.19}$$

Here r_{eq} is the equatorial radius of the Earth and μ is the gravitational constant of the Earth. The variables X, Y, and Z are the inertial position coordinates with respect to the ECI (Earth Centered Inertial) frame and the orbit radius is $r = \sqrt{X^2 + Y^2 + Z^2}$. The magnitude of the J_2 induced relative motion disturbance depends on how the relative orbit

is formed (whether the out-of-plane motion is achieved through inclination or ascending node differences), and on the location (anomaly angle) within the orbit.[5,19,22] Because we are examining relative motion of satellites, we would like to examine the effect that gravitational harmonics have on these relative orbits. A study examined the disturbance acceleration magnitudes for different formation sizes and at different altitudes. In the study, the altitude is swept from LEO (300 to 1000 km) to GEO (35,000 km) and the formation size is swept from 10 – 1000 meters. The results of the study showed that the differential J_2 perturbations increase with increasing formation size and with decreasing orbit altitude. As the chief orbit inclinations are increased, the disturbance accelerations increase slightly, but not substantially.[8] Figure 3.3 shows an example case of the relative motion of 2 craft under the influence of gravitational harmonics. In this plot the same initial separation distance of 200 meters and chief (center of mass) orbit altitude of 300 km are chosen.

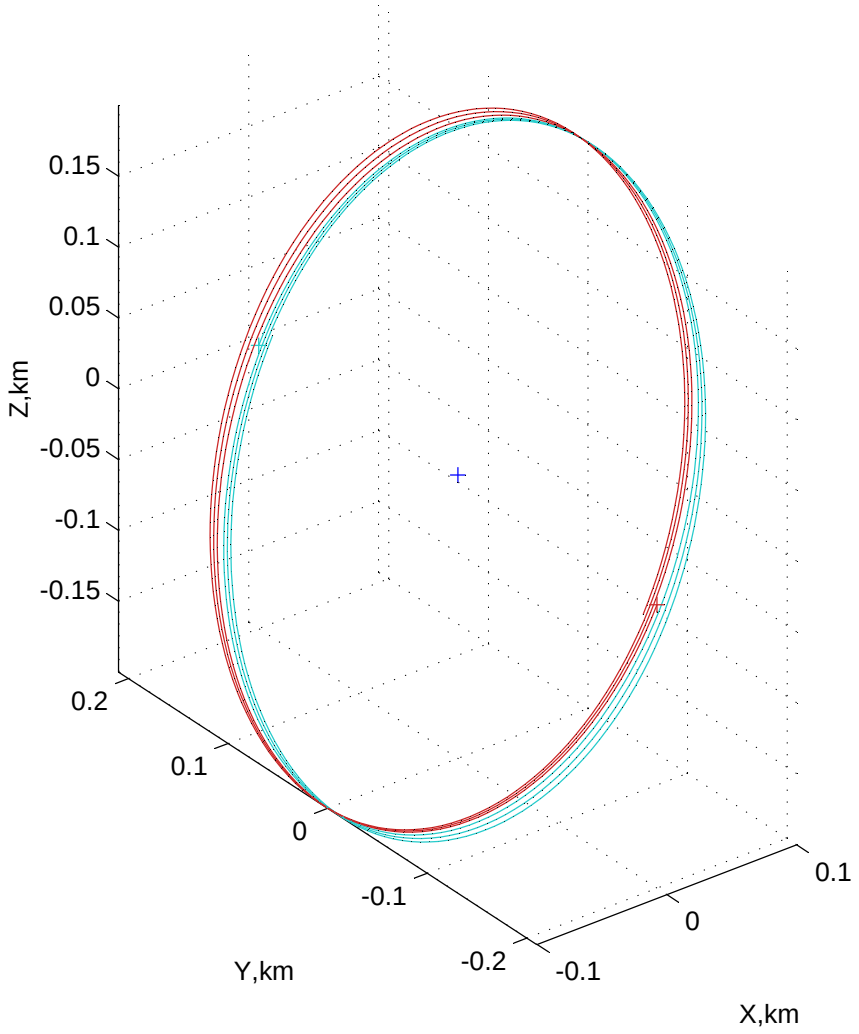


Figure 3.3: Relative Orbit Between 2 Spacecraft with J_2 through J_6 Perturbations

In Figure 3.3, the relative orbits are plotted over 3 orbit periods. Over this time the effect of the J_2 through J_6 perturbations becomes significant, as seen in a shift in the relative orbits along the Y axis. The gravitational harmonics also rotate the relative orbit frame about the along-track and radial directions, which violates the assumptions used in the Hill Frame equations of motion. This illustrates the importance of using the inertial non-linear

equations of motion.

3.3.2 Atmospheric Perturbations

The next perturbation examined is the atmospheric drag force. This effect is strongest in the low earth orbit altitudes and becomes negligible at orbit altitudes greater than 1000 km. The magnitude of acceleration due to atmospheric drag is.[22]

$$a_D = -\frac{1}{2}\rho(C_d A/m)V^2 \tag{3.20}$$

Here ρ is the atmospheric density, C_d is the coefficient of drag, A is the satellite's cross-sectional area, and V is the current inertial velocity of the spacecraft. The drag force acts in the opposite direction of velocity:[22]

$$\boldsymbol{a}_{d_i} = a_{d_i}\hat{\boldsymbol{\imath}}_v = a_{d_i}\frac{\dot{\boldsymbol{r}}_{\boldsymbol{i}}}{|\dot{\boldsymbol{r}}_i|} \tag{3.21}$$

The atmospheric density model used in equation 3.21 is the United States Standard Atmosphere Model from 1976.[18] The actual model contains density data for altitudes ranging from 86 to 1000 km. Above this range it is assumed that density becomes close to zero and the drag force is non-existent.[18] The atmospheric disturbance is not dependent on the spacecraft formation size. Rather, it only depends on the orbit altitude. A plot of two spacecraft in formation under the influence of atmospheric drag is shown in Figure 3.4. A general cylindrical shape is considered for each spacecraft otherwise if spherical spacecraft of equal mass are considered the differential atmospheric draft would be trivially zero. In the cylindrical spacecraft case the attitude of the spacecraft has an influence on the C_d

value. This configuration allows us to study the differential drag force between two or more satellites in a formation. An estimate of the C_d value for worst and best case scenarios is based on existing data from an actual spacecraft.[29] A typical spacecraft with a mass of 50 kg is chosen with a radius of 0.5 meters and a height equal to three times the radius. This provides our numerical simulation with realistic spacecraft parameters.[8]

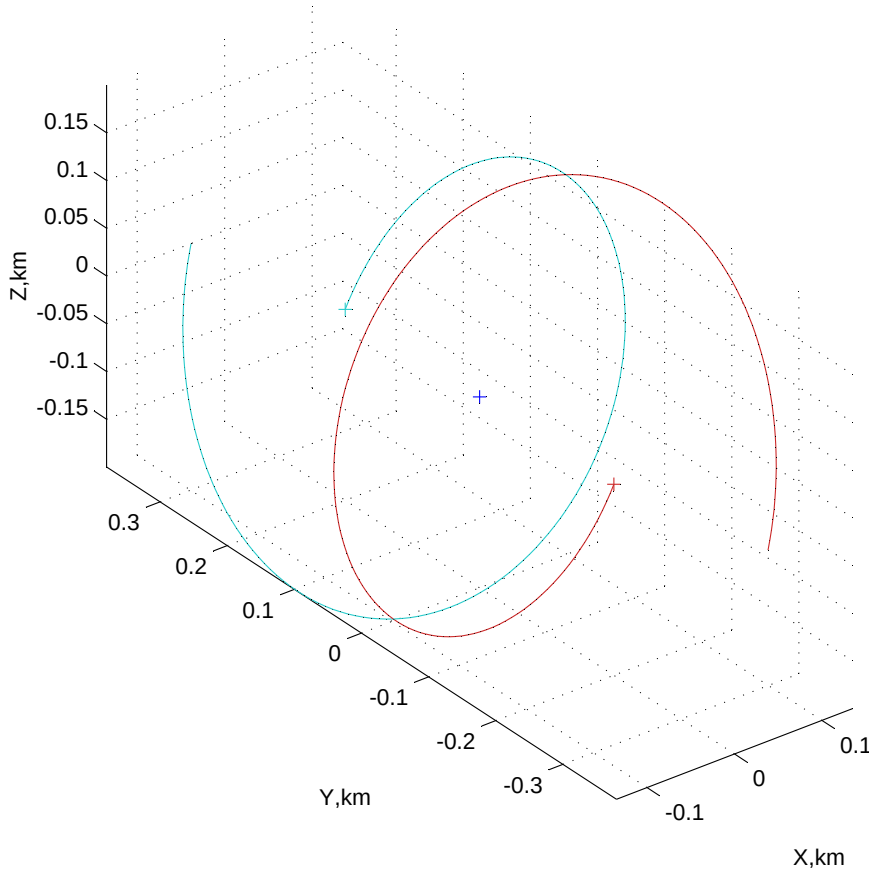


Figure 3.4: Relative Orbit Between 2 Spacecraft with Atmospheric Drag Perturbations

In this case the chief orbit (center of mass) is at 300 km altitude, so the drag effect is relatively high. Also note that we are looking at a worst case scenario in differential drag

force between the two spacecraft. This is why we see them drift apart from each other in the Y-direction. This is due to the fact that the drag force acts along the velocity vector and tends to both circularize and lower the orbit speed.

3.3.3 Solar Radiation Pressure

Solar radiation drag is created by having the sun's light reflect off the spacecraft. Through momentum conservation, a small force is exerted onto the craft. The magnitude of this force depends on the apparent size and reflectivity of the spacecraft. To model the solar radiation pressure a spherical spacecraft model is used. The equation for this model is:[27]

$$\boldsymbol{a}_R = -C_R \frac{A\Phi \boldsymbol{R}}{mcR^3} \tag{3.22}$$

Here A is the cross-sectional area facing the sun, and $\Phi = 1372.5398$ W/m^2 is the solar constant. Further, m is the spacecraft mass, $c = 2.997 * 10^8$ m/s is the speed of light, and C_R is the pressure radiation coefficient. The pressure radiation coefficient is taken to be $C_R = 1.3$ from the average value based on recent data.[27] Lastly, the vector $\boldsymbol{R}$ is the inertial vector pointing from the sun to the planet the spacecraft is orbiting in AU, and R is its magnitude. In this part of the equation, it is assumed that there is a quadratic drop in radiation pressure as the distance is increased past 1 AU. Without loss of generality, for our simulation we choose a vector in the vernal equinox direction as a default. However, the user supplies this distance based on which planet the spacecraft is orbiting. The motion of all satellites is considered to be insignificant compared to the size of this inertial position vector.

The solar radiation pressure does not depend on altitude or formation size.[8] In Figure 3.5, the effects of solar radiation pressure on the spacecraft formation are displayed.

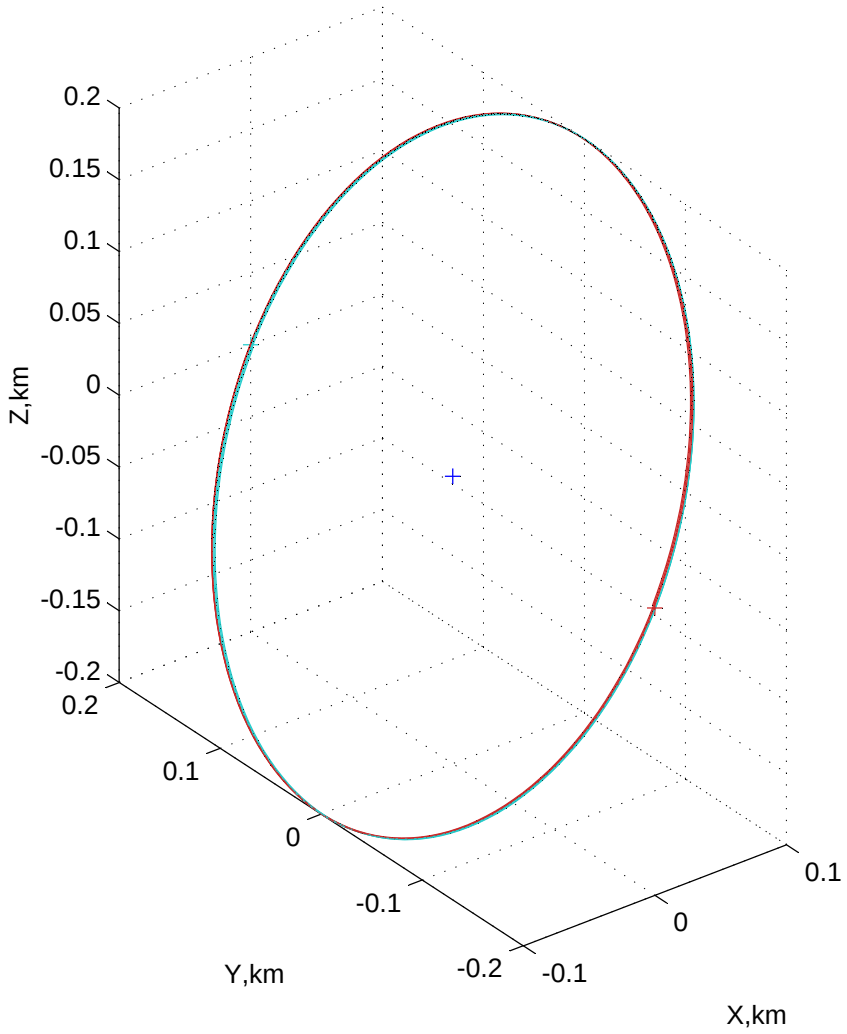


Figure 3.5: Relative Orbit Between 2 Spacecraft with Solar Radiation Pressure Perturbations

Again note that we are looking at the worst case scenario where the differential force is maximum. However, we see that this effect is minimal in comparison to the other perturbations. But, it is still an effect that produces a disturbance acceleration and in combination with the

other effects creates a significant force on the spacecraft. To get an idea of the overall effect of these perturbations, Figure 3.6 provides an overview of altitude and separation distance zones showing which perturbation is the most significant for a particular zone.

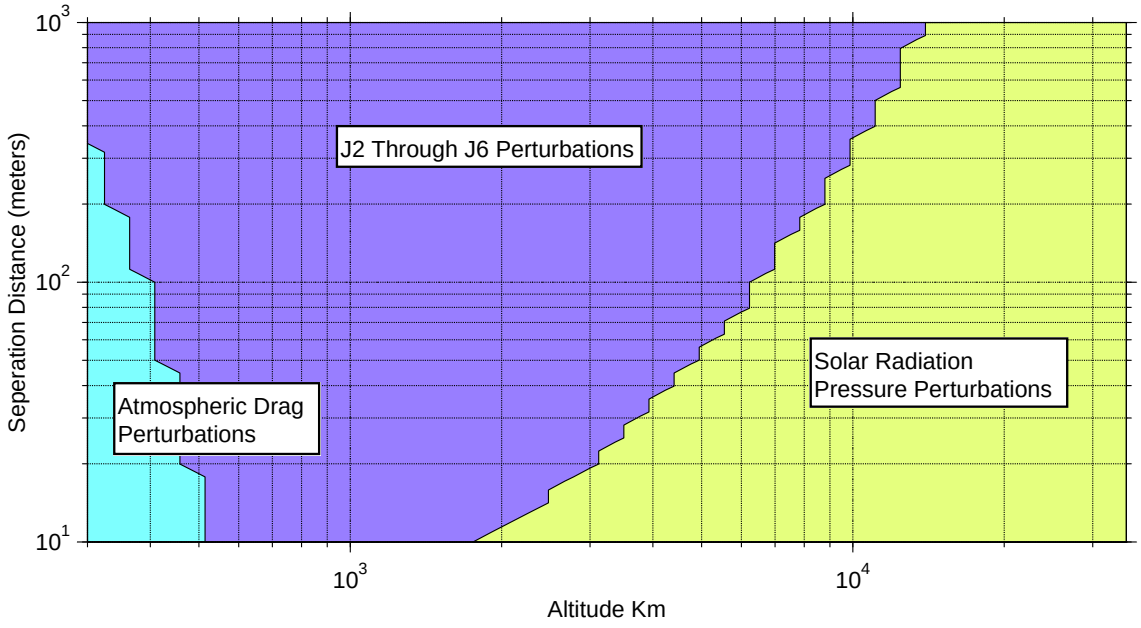


Figure 3.6: Dominant Differential Perturbation Zones Illustration.

Traditional formation flying applications treat the J_2 perturbation as the dominant disturbance of the formation geometry.[3,20,26] Here the spacecraft are assumed to be of equal type and build. However, even if all craft have the same shape, different orientations cause significant differential atmospheric drag in LEO regimes. Figure 3.6 shows that for conditions used in this study, the differential atmospheric drag dominates at LEO up to separation distances of 350 meters. As the orbit altitude is increased to about 500 km, the differential atmospheric drag dominant zones vanish. For large separation distances at LEO the

differential J_2 perturbation becomes dominant, even if differential spacecraft attitudes are considered. This tendency is expected because the differential J_2 perturbation increases with separation distance, while the differential atmospheric drag does not.[8]

Chapter 4

UMBRA Implementation

The goal of this chapter is to illustrate the methodology of implementing these various simulations into the UMBRA framework. The UMBRA framework is a combination of C++ and a scripting language. The scripting language used in the AVS Lab is Tcl/Tk. In the UMBRA framework, the user creates a module of C++ code that interacts with other modules through this scripting language. This feature is achieved by creating input and output connectors that pass data easily from one module to another. It is also possible to pass parameters directly to a module through the use of "wrapper" functions written in C++. This fact makes the UMBRA framework an ideal tool for creating a simulation with many different parts interconnected to each other without having to recompile C++ code. The rest of this chapter is organized into three major sections. The first section deals with implementing the vehicle simulation in such a way that it interacts with other modules in the exact same manner as the hardware does. This allows for easily switching between the

virtual and actual vehicle as well as ensuring the consistency in the module interface. The second section shows how the inertial orbit simulation and relative orbit calculations are implemented. And lastly, an interface module is created to link these these two simulations together. This module essentially allows either the real or simulated vehicle to behave like a spacecraft in a planar orbit.

4.1 The Pioneer Module

The vehicle simulation is based on an existing UMBRA module called pioneer. This module was originally created to control the real robotic vehicle by interfacing with the robot's own C++ libraries called ARIA. These ARIA libraries feature functions that control the vehicles movements, as well as return encoder computed position and heading data. The following section details how the Pioneer module is modified in order to interact with a simulated vehicle, while still keeping the original functionality intact. Figure 4.1 shows a rough schematic of the pioneer simulation and its various components. There are three main components to this simulation. First is the pioneer module itself with the various modifications. Second, is a simulated servo control module which takes commands generated through pioneer and output wheel motor torques. The servo module also serves to simulate other functionalities of the real vehicle that are discussed later. And lastly, a module that encompasses the vehicle dynamics is discussed in Chapter 2.

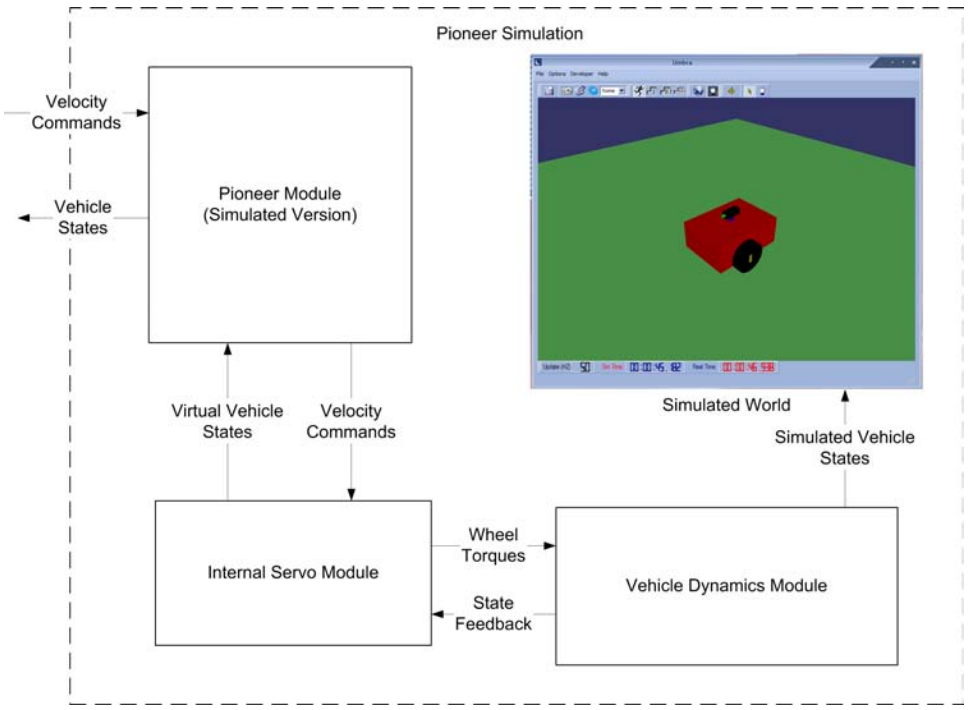


Figure 4.1: Schematic of Pioneer Simulation

4.1.1 The Original Layout

The original pioneer module was created to interface with the ActiveMedia P3-DX Robotics platform. This robotic vehicle is located in the AVS Lab and is used for exploring different aspects of relative motion problems. A diagram of the pioneer module with its various original and simulated connectors is shown in Figure 4.2. The original pioneer module has three input connectors and three output connectors. The first input connector is the `mode` connector, which is an integer between 0 and 3. This connector describes the operating mode of the module. A value of 0 corresponds to no motion at all, and 1, 2, and 3 corresponding to the Teleop, Wander, and Unguarded modes respectively. The second input connector,

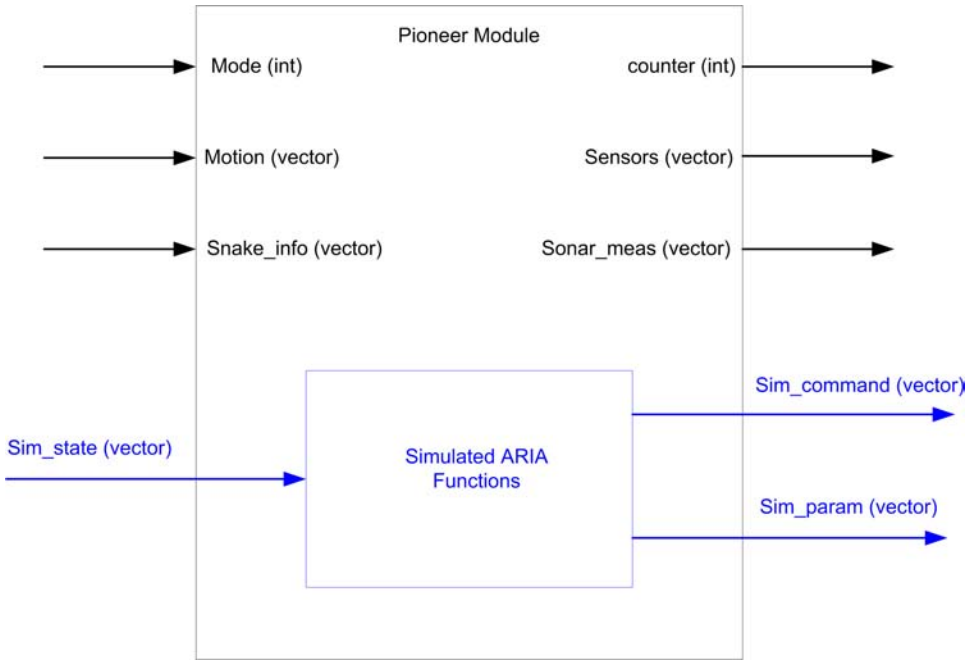


Figure 4.2: Diagram of Pioneer Module

`motion`, is a vector containing four doubles. The `motion` input connector contains different states depending on the mode of the module. The motion input connector components for the Teleop modes are featured in table 4.1. In the Teleop mode there are no direct speed commands, the vehicle only moves in a single direction at the specified maximum rotational or translational velocity. This mode is a useful mode for easily commanding the vehicle to move in different directions but is not used for the simulated version. The wander mode is a purely an ARIA defined mode, where the robotic vehicle moves in randomly picked direction until it comes near to something and then changes direction to avoid hitting the obstacle. The robotic vehicle senses these obstacles using its 16 sonar sensors and various bump sensors. The last mode is the Unguarded mode, which is also the most useful mode. Here, the user

Table 4.1: Motion Input Connector Components for Teleop Modes

	Motion Connector Components	
`[0]`	`[1]`	`[2]`
0 (No Motion)	N/A	N/A
1 (Motion)	± 1 (Move forward or in reverse)	± 1 (Turn right or left)

Table 4.2: Motion Input Connector Components for Unguarded Modes

	Motion Connector Components	
`[0]`	`[1]`	`[2]`
0 (No Motion)	N/A	N/A
1 (Velocity Mode)	V_c (Move at speed V_c in mm/s)	R_c (Turn at speed R_c degrees/s)
2 (Position Mode)	D (Move to distance D in mm)	N/A

controls directly the translational velocity, rotational velocity, or relative position of the vehicle. The motion input connector components for the Unguarded modes are described in Table 4.2. The velocity mode gives the user the most control over the vehicle's movements and is the only mode that is used in this version of the numerical simulation. The last original input connector is a `snake_info` vector, which contains information from a visual snake and is currently not being used. Lastly, there are also three original output connectors which are `counter`, `sonar_meas`, and `sensors`. The first is an update counter to keep track of the number of iterations the module has completed. The second output connector contains

sonar information and is unused currently. The last connector, `sensors`, outputs a vector that contains information on the vehicle's position and heading. These modes are controlled through a Tcl/Tk generated GUI (Graphical User Interface). Figure 4.3 shows screen shots of the GUI in the Teleop and Unguarded modes. The different modes are accessed through a pull down menu of the GUI.

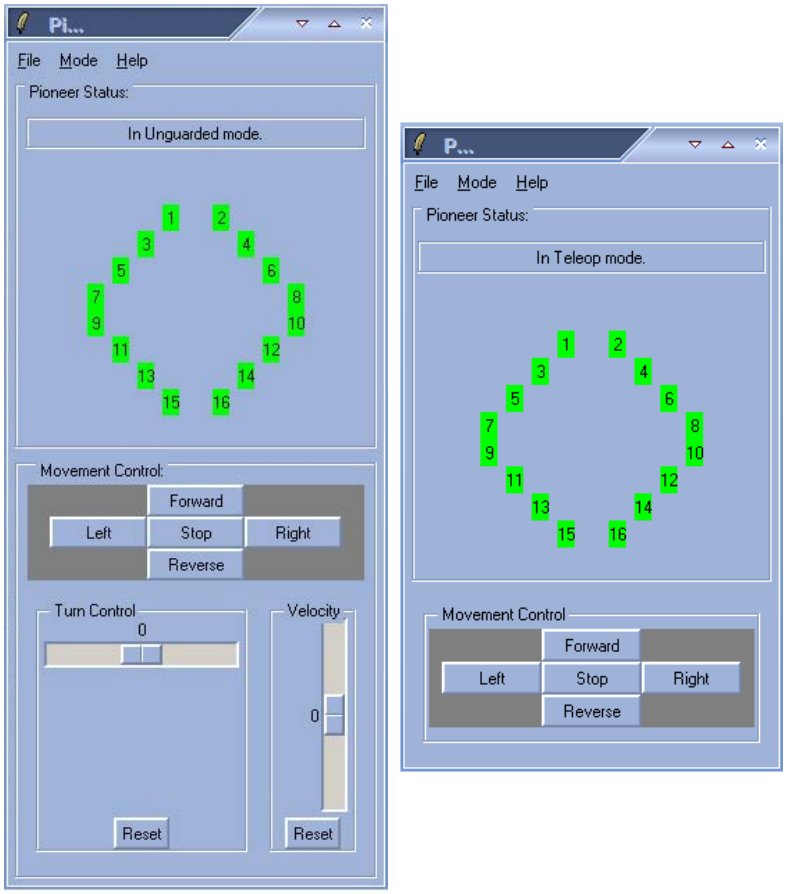


Figure 4.3: Screen Shots of GUI for Unguarded Mode (left) and Teleop Mode (right).

4.1.2 The Modifications

As mentioned before, the only mode of pioneer in which we are interested in is the unguarded velocity mode. This mode is implemented as a separate C++ function as part of the original pioneer module. The implementation of these functions is controlled by an `update()` function. The `update()` function checks the mode, initializes if it is a new mode, and then activates that mode to accept the inputs through the GUI if necessary. Within the Unguarded function are various ARIA functions that command the robotic vehicle. Our goal is to preserve this functionality, while at the same to interface with the virtual vehicle. Table 4.3 features some select functions that control the actual hardware.

Table 4.3: ARIA functions and their purpose.

Function Name	Function Purpose
`robot.setVel()`	Set the translational velocity in mm/s
`robot.setRotVel()`	Set the rotational velocity in $^\circ/s$
`robot.setAbsoluteMaxTransVel()`	Set the maximum translational velocity in mm/s
`robot.setAbsoluteMaxRotVel()`	Set the maximum rotational velocity in $^\circ/s$
`robot.getX()`	Output the estimated X position in m
`robot.getY()`	Output the estimated Y position in m
`robot.getTh()`	Output the estimated heading in $^\circ$

The intention for these modifications is to keep these functions intact and instead use them to pass various data to and from the simulated vehicle. This goal is achieved by circumventing the provided ARIA libraries and defining our own simulated versions of these functions. These functions are defined in a header file *AriaSim.h* and essentially replace the actual ARIA libraries. This file is included in the main module file instead of the actual libraries. Through these functions information is passed through additional "simulated" connectors which are shown in blue in Figure 4.2. An additional header file, *AriaSimfun.h*, contains the function definitions that pass data to various simulated connectors. First, the input connector `sim_state` contains the vehicle states from the numerical simulation which is then obtained by the pioneer module through the `robot.getX()`, `robot.getY()`, and `robot.getTh()` functions. These functions output the vehicle's x and y position as well as heading angle, respectively. An output connector `sim_param` contains the maximum allowable rotational and translational velocity, which are the speed limits for the vehicle.

The next output connector is `sim_command`, which is a vector containing six numbers. The first component is either 1, 2, or 3 which tells a connecting module what mode to be in and what type of information to expect in the remaining components of the vector. Currently only mode 2 is operational, which is the unguarded velocity mode. In this mode, the second and third components of `sim_command` are the translational velocity and rotational velocity, which are passed through the simulated functions `robot.setVel()` and `robot.setRotVel()`.

4.2 The Full Vehicle Simulation

Figure 4.4 shows a block diagram of all of the different modules connected together to form the complete pioneer vehicle simulation. The basic idea is that the pioneer module generates commands through its GUI or by interfacing with another simulation and passes them along to the simulated servo module. In turn, the servo module calculates the wheel torques and passes them along to the vehicle dynamics. The cart dynamics module takes the torques and the previous state and calculates the derivative of the state, which is sent to an integrator module to calculate the new state and feed it back. The cart dynamics module contains the kinetic equations of motion for the virtual vehicle discussed in Chapter 2. The integrator in this case is a simple Euler integrator. This procedure runs every time UMBRA updates the modules. Also note that the simulation runs in either real or simulated time, which is set by a Tcl procedure called `SetTime`. In this function, 0 corresponds to real time and a non zero value corresponds to the new time step of the simulation. Since we have preserved the functionality and layout of the original pioneer module, we are able to interface with any code or other modules written for the original pioneer module without having to make any changes.

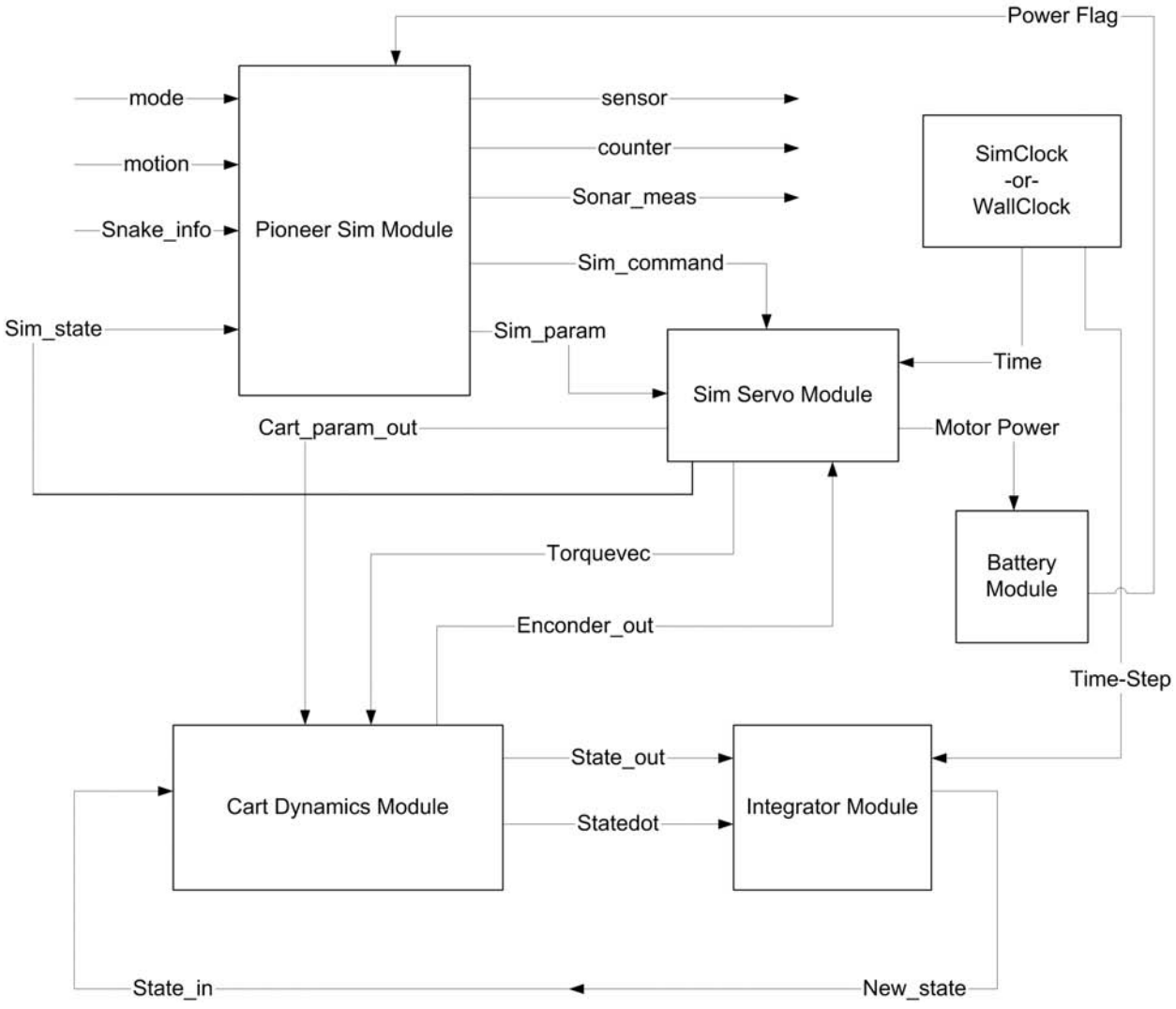


Figure 4.4: Block Diagram of Pioneer Simulation

4.3 Servo Module

The goal of the simulated servo module is to create an interface between the pioneer module and the vehicle dynamics modules. The servo module's main task is to generate left and right wheel torques from velocity or position commands. It has additional tasks such as reading a simulated encoder output and estimating the state of the vehicle and simulating battery power consumption. The next section details the layout of the simulated servo module.

4.3.1 Layout

The servo module has a total of five input connectors and four output connectors that interface with other modules of this simulation. The first two input connectors, `command` and `param`, connect with the two simulated output connectors of the Pioneer module, `sim_command` and `sim_param`. These connectors allow the Pioneer module to pass along any commands and parameters to the servo module. The input connectors `time` and `time_step` interface with UMBRA's wallClock or simClock modules to add real time or simulated time. These modules are standard UMBRA modules that are initialized upon running the UMBRA program. The last input connector, `encoder_out`, contains a simulated wheel encoder output from the dynamics module that is used for state estimation. The `Torquevec` output connector contains the left and right wheel torques and passes them to the vehicle dynamics module. The second output connector, `cart_param_out`, contains the parameters of the robotic vehicle, such as wheel base and wheel radius, which are set internally in the servo

module and passed along to the dynamics modules. These connectors ensure consistency of parameters across all the modules. The last output connector is the `sim_state` connecter which is in a feedback connection to pioneer module. This connector allows pioneer to obtain the state of the vehicle through the simulated ARIA functions. The layout of the servo module is featured in Figure 4.5 along with some descriptions of the internal logic and functions.

4.3.2 Digital Filtering and Differentiation

The simulated servo module contains functions that represent the different states of the servo. There are three main functions in the module which correspond to the three movement modes of pioneer: unguarded velocity, unguarded relative position, and Teleop modes. The unguarded relative position and Teleop modes will be developed in a later version of this module. For the first iteration of the servo module, the unguarded velocity function is created and is represented as `UngaurdedVel()` in Figure 4.5. In this function, there are two inputs: the rotational velocity and translational velocity. These two terms are independent of each other. For example, the vehicle moves forward at a given velocity and at the same time rotates at a given heading rate and the resulting motion is turning in an arc. It is necessary to obtain the rotational and translational accelerations in order to compute the necessary torque to apply to the wheels. To achieve this, we take advantage of a 1st-order differentiator with a 1st-order low-pass filter. The additional filter helps remove any noise and smooth any discretization from the velocity inputs. The differentiator with filter is

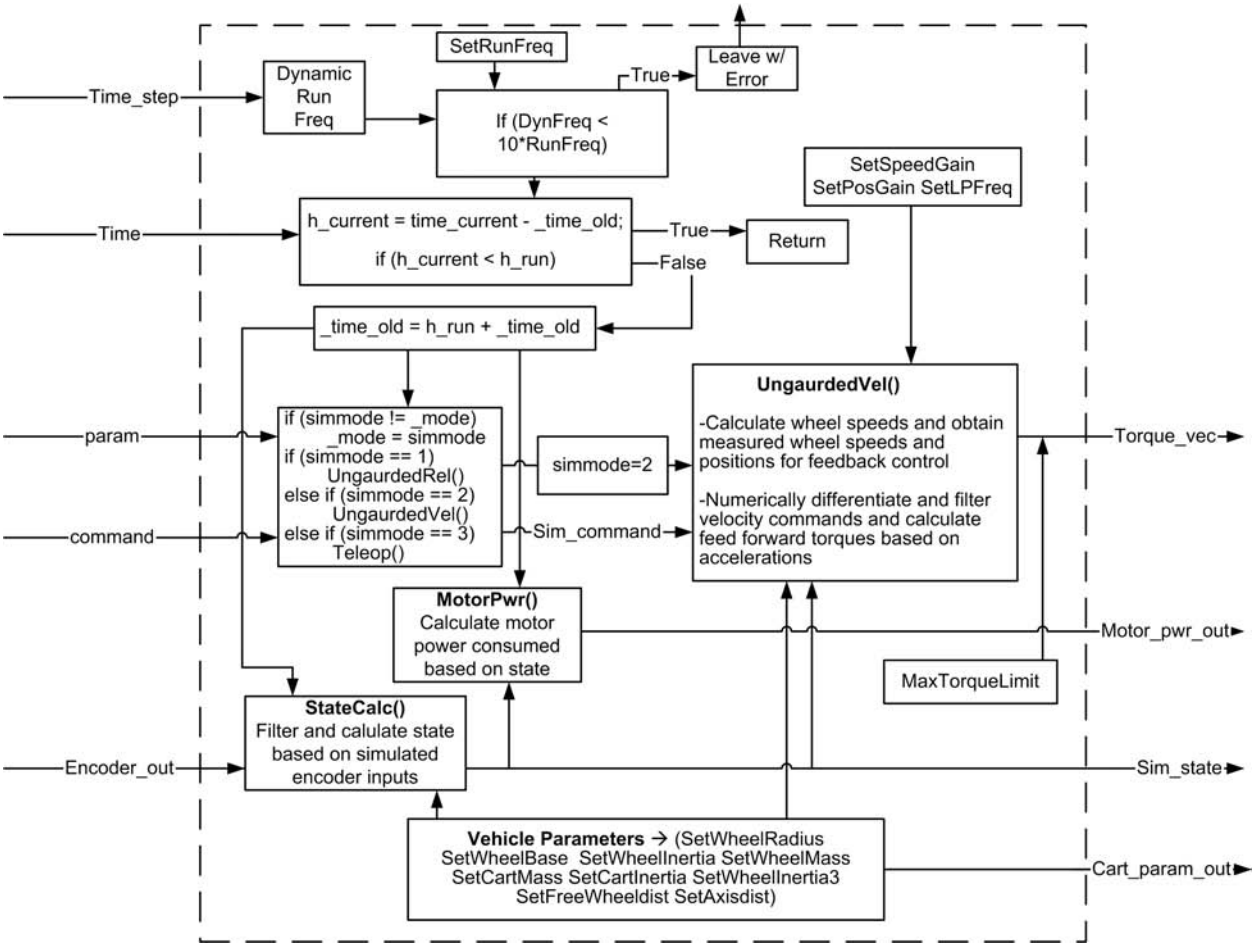
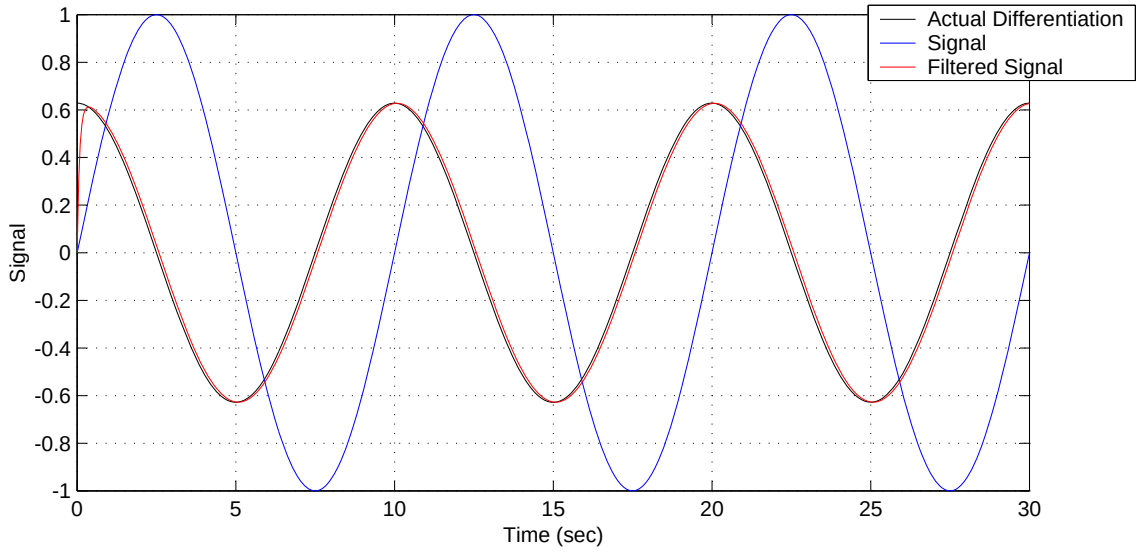


Figure 4.5: Diagram of the Servo Module.

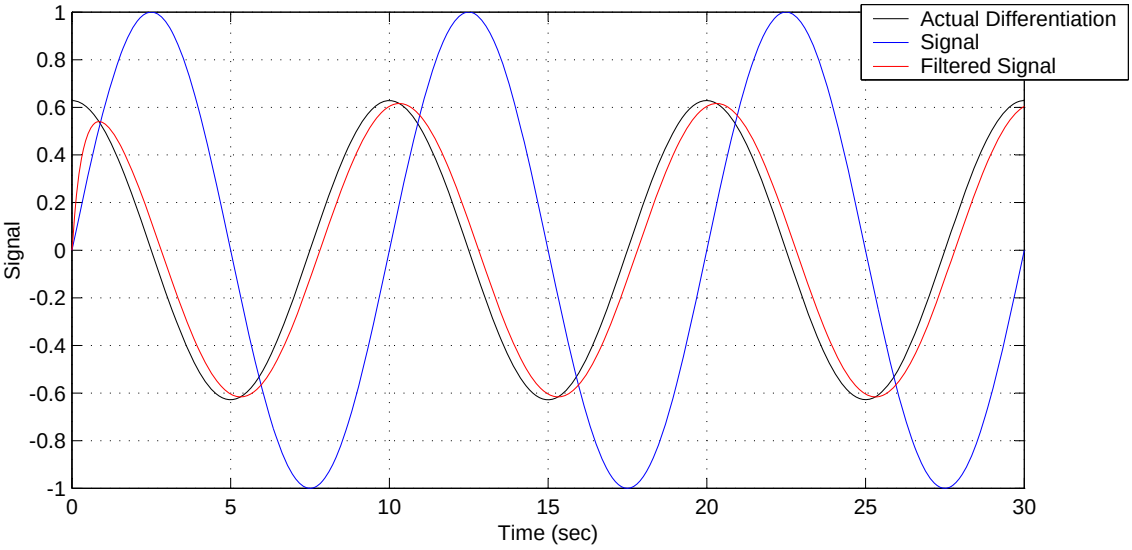
featured in equation 4.1.[11]

$$a_i = \frac{1}{2 + h\omega_c}[a_{i-1}(2 - h\omega_c) + 2\omega_c(v_i - v_{i-1})] \tag{4.1}$$

In this equation, a_i and a_{i-1} represent the current and previous accelerations, while v_i and v_{i-1} are the current and previous velocities. Also, h and ω_c are the time step and cut-off frequency in rad/s. In order to get an estimate on the behavior of this differentiator and select a cut-off frequency, a sinusoidal signal running at 0.1 Hz is filtered and differentiated with and without noise. For our noise model, an approximate Gaussian distribution is chosen. The sinusoid motion is chosen based on the typical vehicle commands. Figure 4.6 shows two plots of the output signal with two different cut-off frequencies. From the figures we see that when the cut-off frequency is at 0.5 Hz, we get some lag in the signal. On the other hand, a higher cut-off frequency that is at least 10 times higher than the signal frequency greatly reduces this effect. The drawback is that if there is any noise below 2 Hz it is not smoothed out and even grows larger after differentiation. In Figure 4.6 some Gaussian noise is introduced. As expected, differentiating only magnifies this effect, unless the cut-off frequency is set to a lower value. In the figure, a cut-frequency of 0.5 Hz is used and the noise remains reasonably low with a small amount of lag. This differentiator is implemented as a separate function in the main module. The next problem is applying this filter to the servo module. UMBRA modules update as fast as possible according to the computer's limits. However, the servo module must update slower than the vehicle dynamics module. Otherwise, the control may be updating faster than the dynamical system. Ideally, the servo module is to run 5 times slower than the dynamics module. This check is done using an if

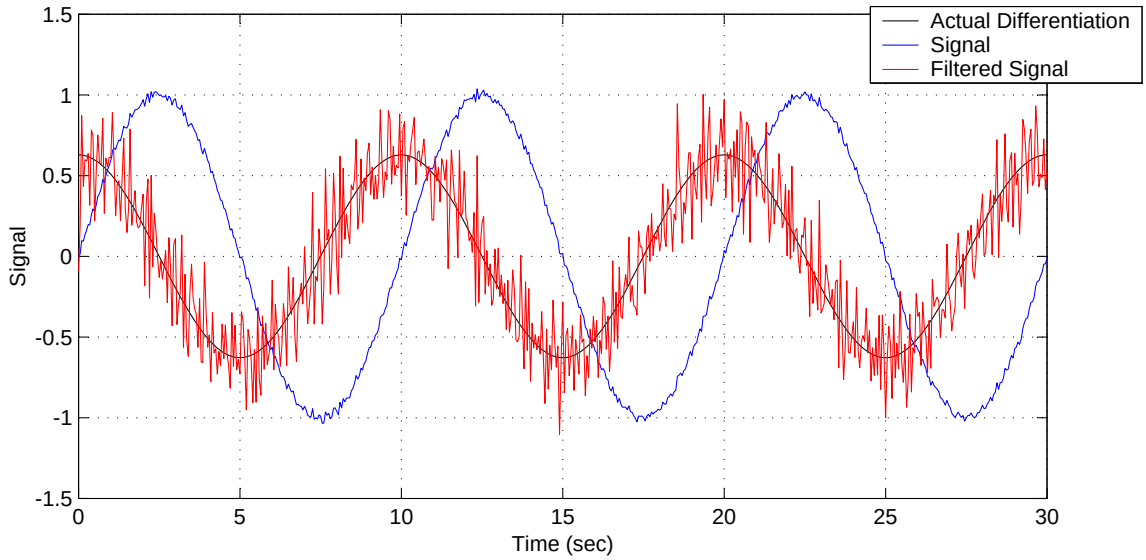


(a) Plot of Filtered Signal with cut-off frequency at 2 Hz

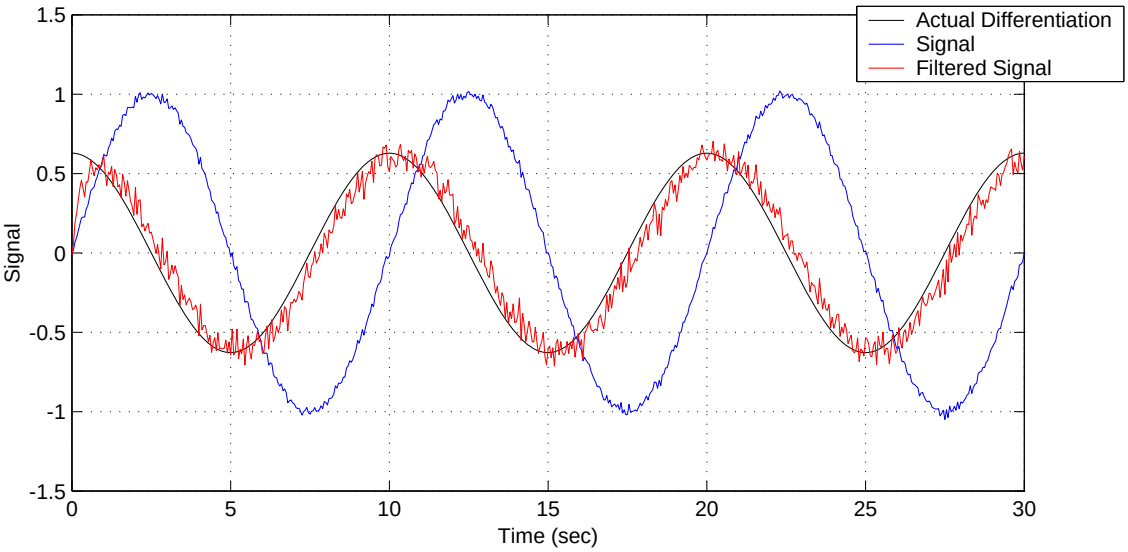


(b) Plot of Filtered Signal with cut-off frequency at 0.5 Hz

Figure 4.6: Filtered Response to Sinusoidal Signal.



(a) Plot of Filtered Signal with cut-off frequency at 2 Hz



(b) Plot of Filtered Signal with cut-off frequency at 0.5 Hz

Figure 4.7: Filtered Response to Sinusoidal Signal with Noise

statement that is featured at the top of Figure 4.5. To implement the frequency enforcement, the `update()` function is encompassed with a series of if statements that force the module to only update at a certain rate by keeping track of the real time passed between updates. This feature of the servo module is shown in upper portion of the module diagram in Figure 4.5. For our first implementation we chose a run frequency of about 10 Hz. However, both the run frequency and the cut-off frequency of the low pass filter, are able to change in real time through the Tcl/Tk functions `setRunFreq` and `setLPFreq`. These set functions makes it easier to change the filtering properties for different situations.

4.3.3 Computing Wheel Torques with Feed Back

The next key component is to compute wheel torques based on the translational and rotational acceleration commands, which are outputs of the differentiator/filter function. The accelerations are represented as a_r and a_t in Figure 4.8. There are two main parts to computing wheel torques. First, a simplified version of the equations of motion is used to generate a reasonably accurate feed-forward torque term. Secondly, a feed-back term is added based on the desired angular wheel speed and position. The feed-forward term itself has two components. First is a torque that is applied to both wheels equally to provide forward motion and the second is a torque difference that rotates the vehicle. The feed-forward torques are represented in the following equation.

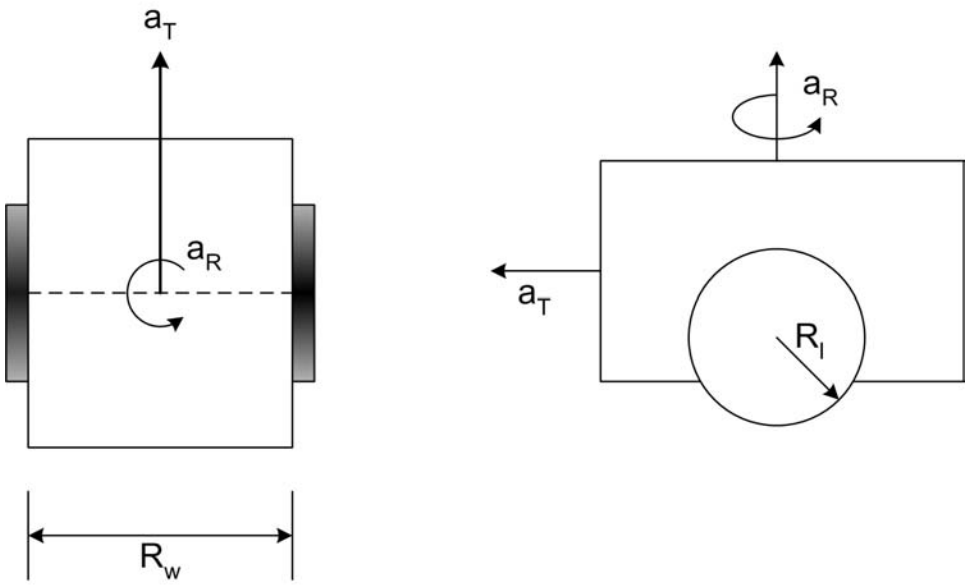


Figure 4.8: Diagram of robotic vehicle.

$$\tilde{T}_R = T_f + \Delta T/2 \tag{4.2}$$

$$\tilde{T}_L = T_f - \Delta T/2 \tag{4.3}$$

To compute the forward torque, T_f, the equations of motion are reduces to a simpler form by assuming only translational motion (θ=0, $\dot{\theta}$=0, $\dot{x}$=0, $T_R = T_L = 2T_f$, $\ddot{y} = a_t$) to obtain:

$$(2(L^2 m + I_{w3}) + I_3)\ddot{\theta} + \frac{I_{w1}L}{R}\dot{\omega}_R - \frac{I_{w1}L}{R}\dot{\omega}_R = 0 \tag{4.4}$$

$$(2m_w + m_c)\ddot{x} = 0 \tag{4.5}$$

$$(2m_w + m_c)\ddot{y} + \frac{I_{w1}}{R}\dot{\omega}_R + \frac{I_{w1}}{R}\dot{\omega}_R = (T_R + T_L)/R \tag{4.6}$$

$$\ddot{y} = \frac{R}{2}(\dot{\omega}_R + \dot{\omega}_L) \tag{4.7}$$

$$\ddot{\theta} - \frac{R}{2L}\dot{\omega}_R + \frac{R}{2L}\dot{\omega}_L = 0 \tag{4.8}$$

Using simple algebra, the forward wheel torque is determined in terms of an translational acceleration a_t:

$$T_f = \frac{(R(2m_w + m_c) + 2I_{w1}/R)}{2} a_t \tag{4.9}$$

For rotational motion a similar process is applied to the equations of motion ($\dot{x} = 0$, $\dot{y} = 0$, $\ddot{\theta} = a_r$, $\Delta T = T_R - T_L$) and the torque difference is obtained in terms of rotational acceleration:

$$\Delta T = \left(\frac{(2(L^2 m + I_{w3}) + I_3)R}{L} + \frac{2I_{w1}L}{R} \right) a_r \tag{4.10}$$

These two equations are used with equations 4.2 and 4.3 to compute a feed forward torque input for each wheel.

However, it is also desirable to include some feed-back on wheel position and rate. The feed-back control enables the virtual vehicle to track the commanded velocities more accurately. Additionally, the wheel angle feed-back ensures that the vehicle is at the correct desired position. For example, the kinetic model does not respond immediately to velocity commands due to its weight and inertia, therefore the feed-back is necessary to compensate for this effect. The feed-back control law is:

$$T_i = \tilde{T}_i + k_v(\omega_{id} - \omega_{im}) + k_p(\theta_{id} - \theta_{im}) \tag{4.11}$$

Here the d subscript represents a desired state and the m subscript represents a measured or estimated state. The desired wheel speeds and positions are computed easily from kinematics

in terms of the original velocity commands.

$$\omega_{id} = \frac{V_t}{R} \pm \frac{\omega_{diff}}{2} \tag{4.12}$$

$$\omega_{diff} = \frac{V_r 2L}{R} \tag{4.13}$$

$$\theta_{id} = \omega_{id} h + \theta_{iold} \tag{4.14}$$

Here V_t and V_r are the translational and rotational velocity commands respectively. Also note that θ_{id} is computed using a simple integration with time step h.

Next, the stability of the closed-loop dynamics with this control law is examined. The closed-loop dynamics equation is obtained by substituting the control torques into the simplified equations of motion featured in equation 4.4. This yields one independent second order differential equation for each wheel which are written as follows.

$$\left(\frac{m_t R}{2} + \frac{I_{w1}}{R}\right) \delta\ddot{\theta}_i + k_v \delta\dot{\theta}_i + k_p \delta\theta = 0 \tag{4.15}$$

Here, $\delta\ddot{\theta}_i = \delta\dot{\omega}_i$ and $\delta\dot{\theta}_i = \delta\omega_i$. This equation represents a spring-mass-damper system, where positive gains k_v and k_p yield a stable behavior and the error in wheel position and speed is driven to zero. In the full torque computation the gains k_p and k_v are selected to mimic the motion of the actual vehicle or of a different vehicle. An analysis of the gain selection is done in a later section.

4.3.4 State Estimation

Another component of the simulated servo module is to estimate the state of the robotic vehicle in the same manner as the real vehicle. One method in which the state is estimated is through what are called encoder outputs. An encoder is a device that reads ticks which are distributed around the wheel, allowing the vehicle to measure the wheel position. These are differentiated to get wheel speed. The P3-DX has 2000 ticks on a wheel and the relationship between wheel angle and encoder tick is:

$$enc_i = \theta_i \frac{2000}{2\pi} \tag{4.16}$$

This equation is implemented in a separate function within the vehicle dynamics module. The encoder output is then fed back to the servo module where the state is estimated. One important issue with using the encoder output enc_i is that it is an integer, therefore the wheel angle is discretized and not a smooth function. Thus, differentiating the encoder output creates noise in the signal. Therefore, a low-pass filter is implemented to smooth the encoder outputs. This filter is the same that used for filtering velocity commands. The

vehicle states are now calculated using simple kinematics:

$$\theta_i = (2\pi enc_i)/2000 \tag{4.17}$$

$$\delta\theta_i = \theta_i - \theta_{iold} \tag{4.18}$$

$$\theta = \theta_{old} + (\delta\theta_R R - \delta\theta_L R)/2L \tag{4.19}$$

$$y = y_{old} + (\delta\theta_r R\cos(\theta) + \delta\theta_l R\cos(\theta))/2 \tag{4.20}$$

$$x = x_{old} - (\delta\theta_r R\sin(\theta) + \delta\theta_l R\sin(\theta))/2 \tag{4.21}$$

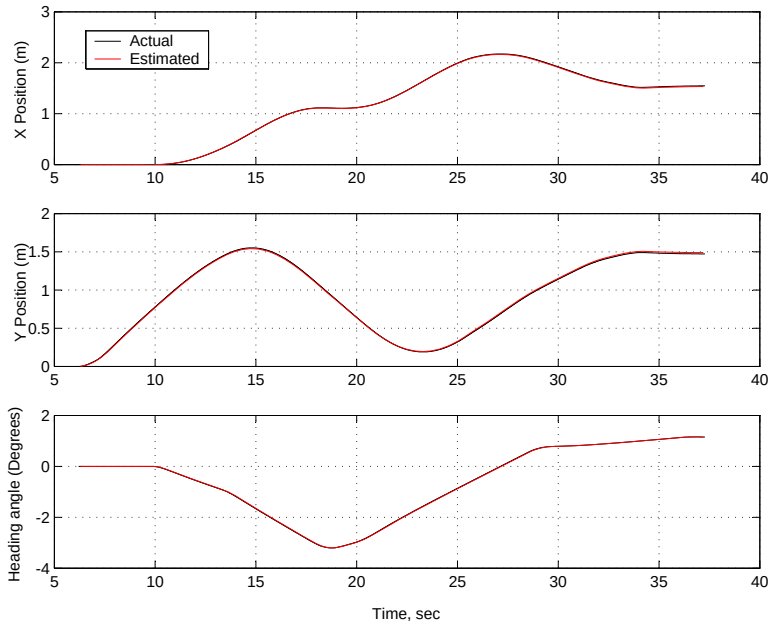
$$\dot{x} = (x - x_{old})/h \tag{4.22}$$

$$\dot{y} = (y - y_{old})/h \tag{4.23}$$

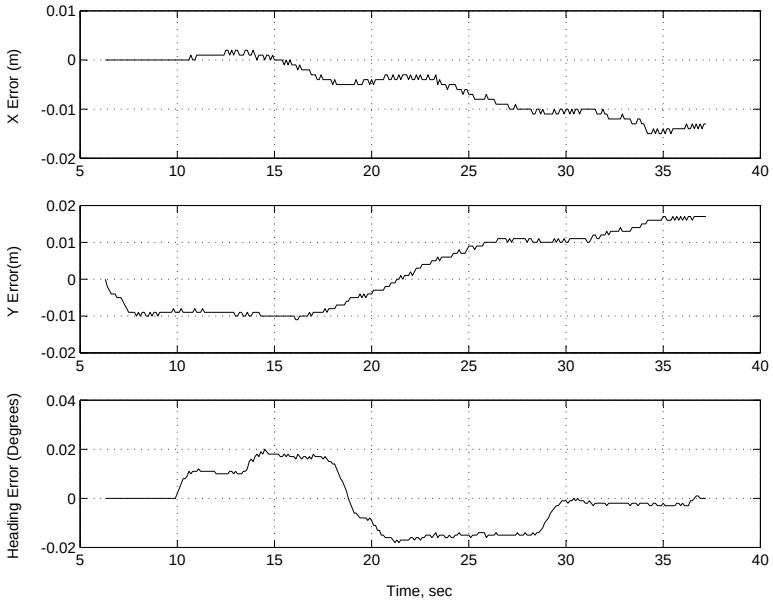
$$\dot{\theta} = (\theta - \theta_{old})/h \tag{4.24}$$

$$\omega_i = \delta\theta_i/h \tag{4.25}$$

In these equations θ_i represents the wheel angles and h is the time step. Also note that the *old* subscript represents the state at the previous update, which is stored in a global variable. The next step is to verify that these equations estimate the state correctly. To achieve this goal, the estimated state is plotted versus the actual state from the vehicle dynamics. In these cases, a cut-off frequency of 5 Hz is chosen, which smooths the encoder outputs reasonably well. Also, the vehicle is controlled manually using the GUI in unguarded mode and allowed to follow some randomly selected path. Additionally, note that the sampling rate of the estimated state is 10 Hz, which is the update frequency of the servo module. From Figures 4.9 and 4.10 we observe that the estimated state follows the actual state closely with minimal noise in the velocity and relatively small error. Additionally, if the filter frequency is lowered

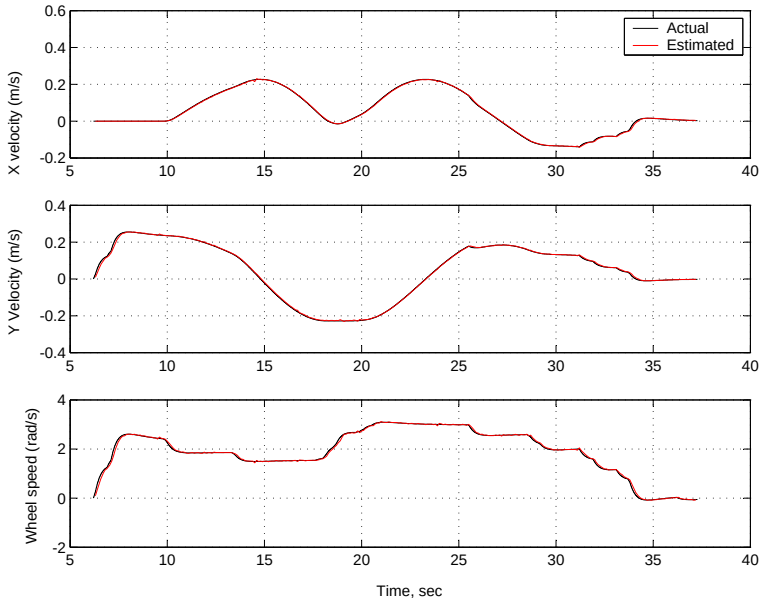


(a) Position Estimation

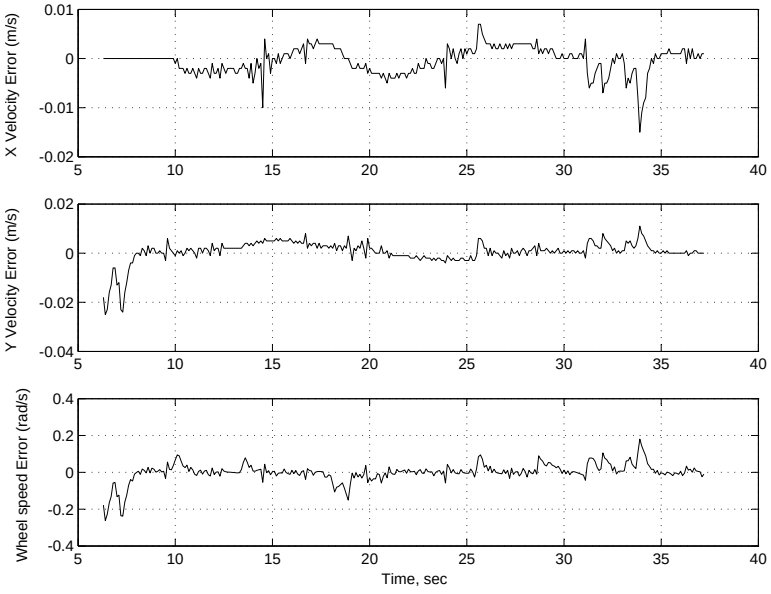


(b) Position Error

Figure 4.9: Position Estimation with LP frequency of 5 Hz



(a) Velocity Estimation



(b) Velocity Error

Figure 4.10: Velocity Estimation with LP frequency of 5 Hz

below 5 Hz the estimated state tends to lag. If the frequency is increased much past this value, noise begins to enter the signal, especially in the velocity estimation. Therefore, 5 Hz is chosen as the frequency for this filter and can be changed by the user if necessary using the Tcl function `SetEncFreq`.

4.3.5 Torque Input Gain Selection

Now that the state estimation has been verified, we use that estimated state to drive the feed-back control on the torque inputs. More specifically, in relation to equation 4.11, control the vehicle using both wheel angle and wheel position. There are two different aspects to selecting gains that must be addressed. The gains are selected such that the virtual vehicle actually performs better than the real vehicle, or these gains are selected to mimic the real vehicle. For the purposes of this thesis, the gains are selected to mimic the real vehicle. The real vehicle exhibits a small delay between commanding the vehicle to stop and achieving the stopped condition. This observation means that the actual internal servo is not using wheel angle or position in the feed-back control. If it were, the feedback control would actually cause the vehicle to go in reverse when it overshoots the desired position. Since this does not happen, the position gain, k_p is picked to be zero. However, the vehicle is controlled based on commanded wheel speed, which would ensure that the vehicle is traveling at the commanded speed. Therefore, our goal is to pick a gain such that the commanded wheel speeds match reasonably well with the measured wheel speeds. Figure 4.11 shows an output of commanded *vs* measured wheel speed with the gain set to zero. Note that in the following

figures the vehicle is controlled manually through the GUI, which is why the commanded wheel speeds are discretized. In Figure 4.11, there is no feed-back control and the wheel speeds exhibit an over-damped behavior and lag behind the commanded wheel speeds. When the speed gain is increased to 0.1 in Figure 4.12, an under-damped behavior is seen. The wheel speeds overshoot and slowly oscillate toward the desired speed. Therefore, the gain is lowered to about 0.03. Figure 4.13 shows an essentially critically-damped behavior with a small amount of lag between the two. This lag is smaller than with having no feed-back, as seen in Figure 4.11, but is reasonably consistent with the actual vehicle's behavior. However, note that these gains can be selected according to the users desired closed-loop behavior. As a result of these feed-back gain selections, controlling the virtual vehicle has a similar feel to controlling the actual P3-DX robotic vehicle.

4.4 Battery Power Module

Another aspect of the vehicle that we desire to model is the battery power consumption. The robotic vehicle has an on-board PC-104 computer and an electric motor for each of the drive wheels. These are powered by a series of lead-acid batteries. Our first goal is to model the battery power and its consumption in the robotic vehicle. Currently, there are numerous battery consumption models, that use factors such as temperature and cycles of use. For a first iteration a simple model derived from basic physics is used. A battery has two properties that describe its performance: its voltage rating and capacity, in Volts and

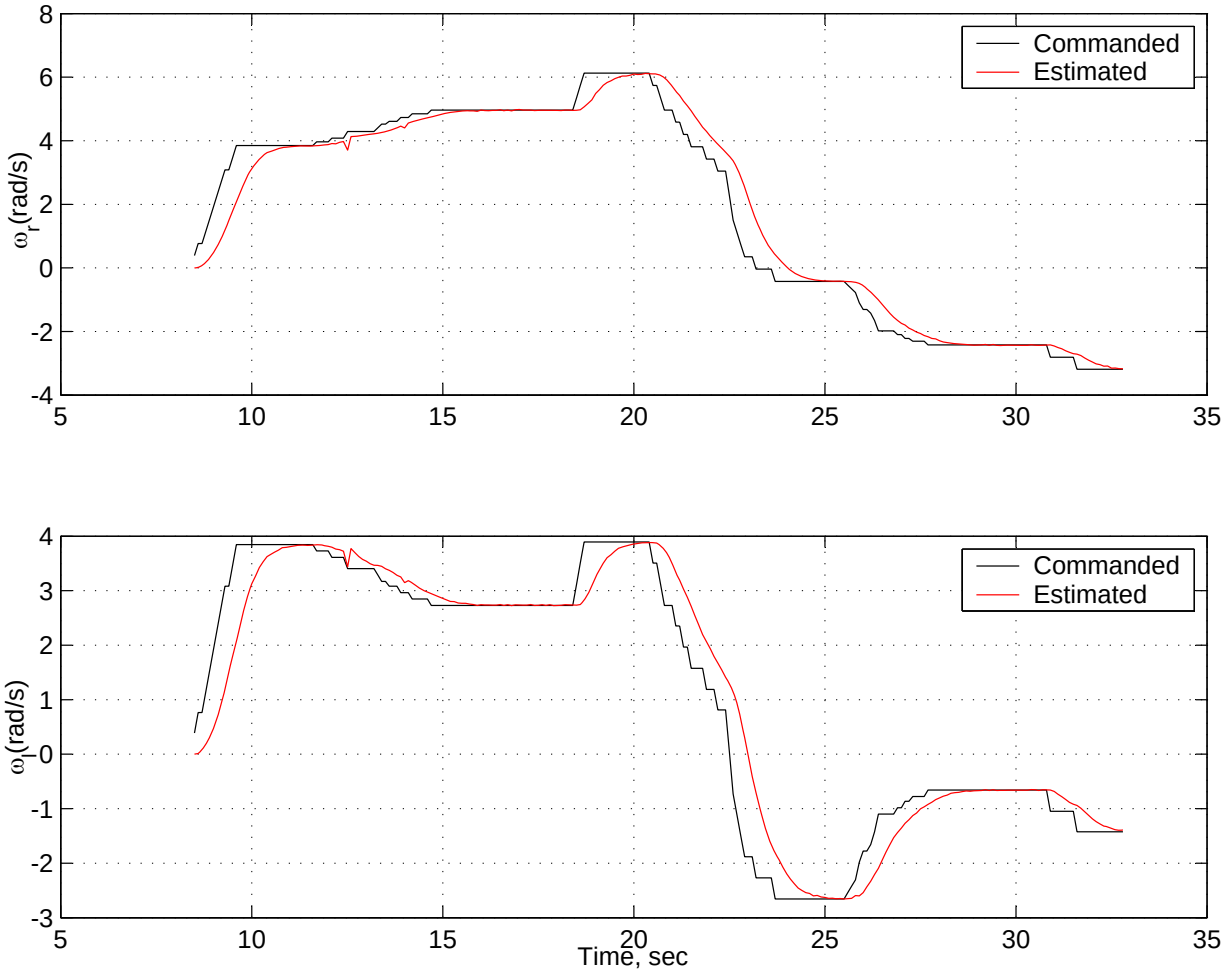


Figure 4.11: Command vs. Estimated with $k_v = 0.0$

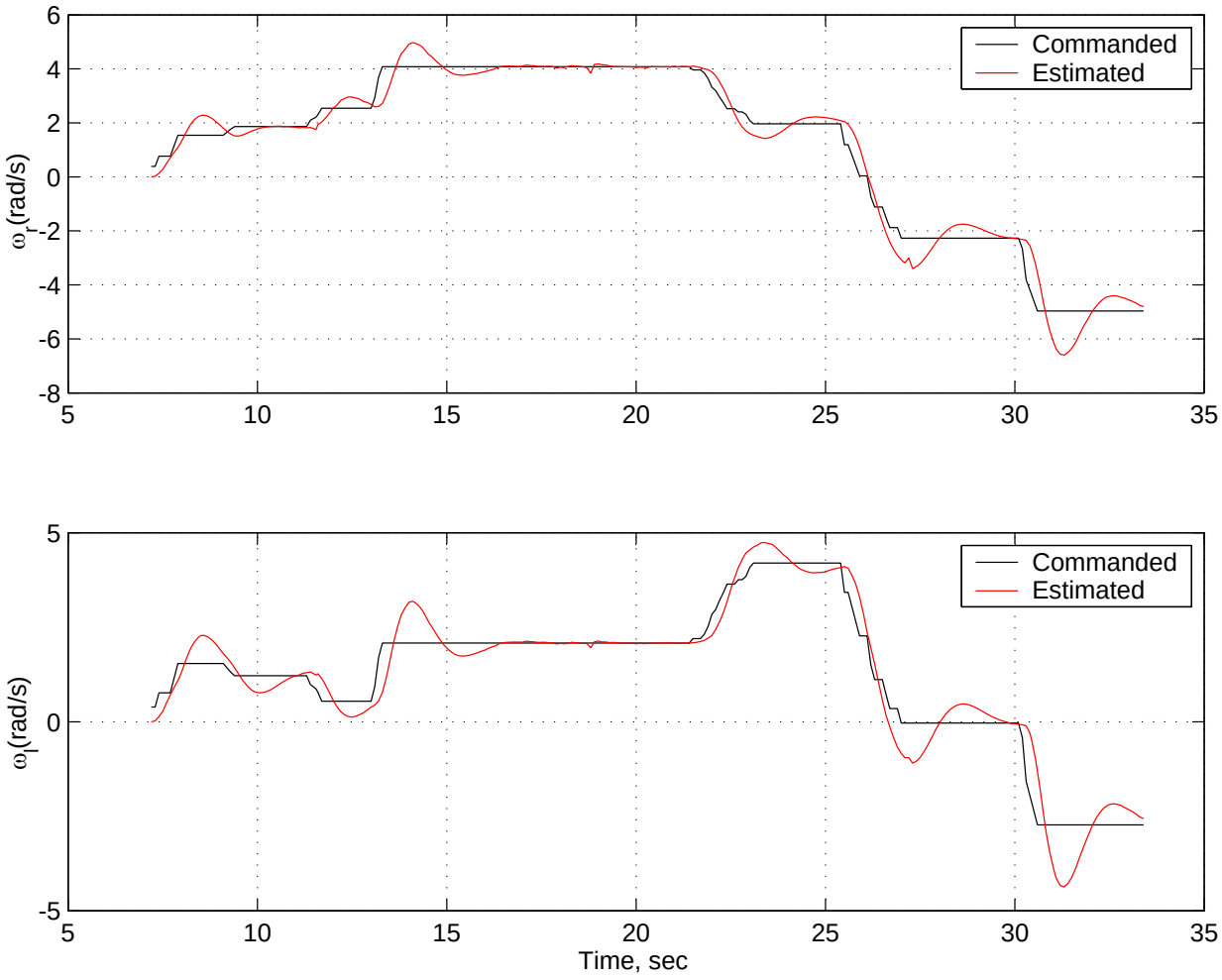


Figure 4.12: Command vs. Estimated with $k_v = 0.1$

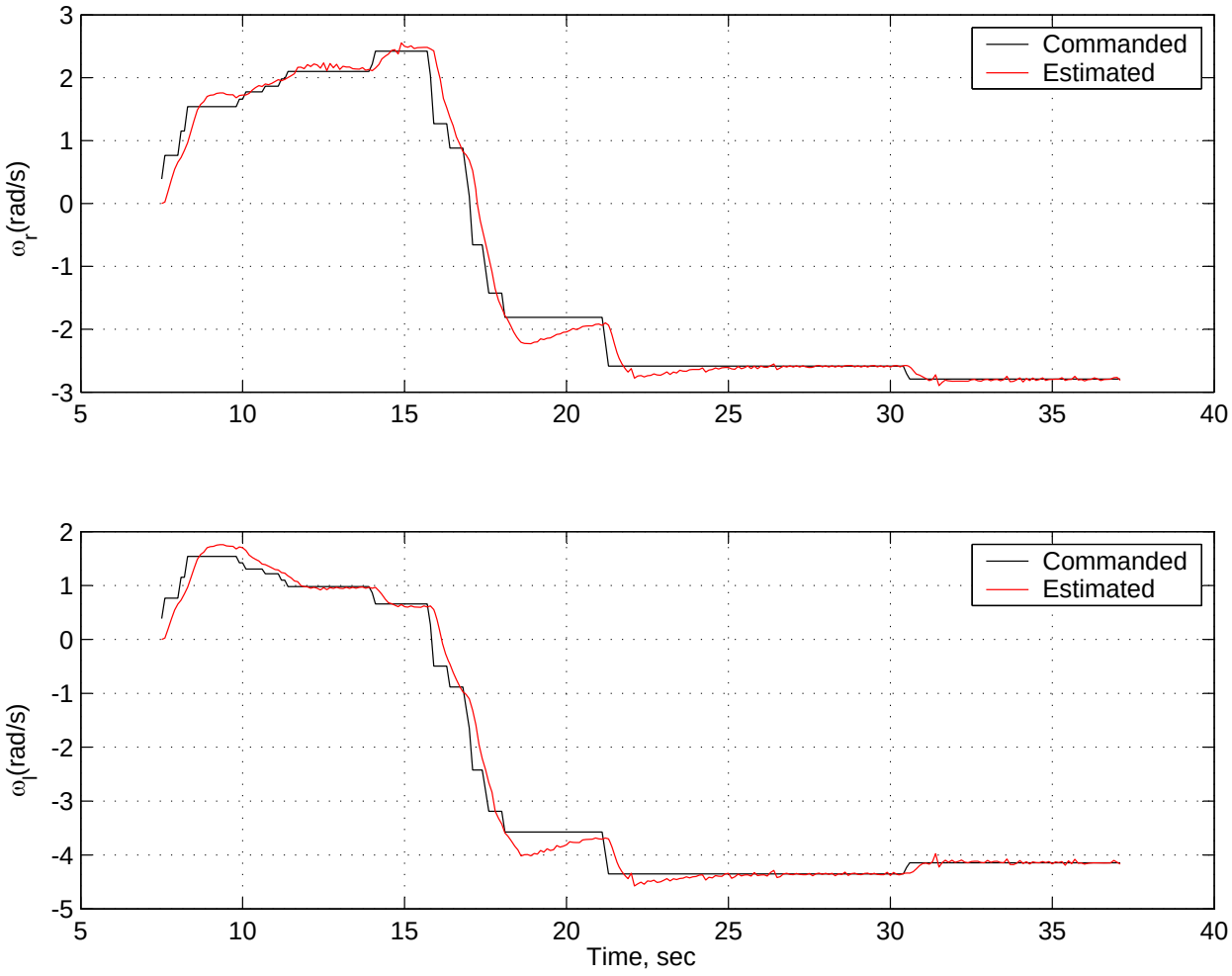


Figure 4.13: Command vs. Estimated with $k_v = 0.03$

Ampere-Hours respectively. The amount of energy in the battery at full charge is illustrated in equation 4.26.

$$Energy(W \cdot h) = VoltageRating(V) \times Capacity(A \cdot h) \tag{4.26}$$

This value represents an estimate of the amount of energy in the battery at full charge. The power draw of the electric motors must also be modeled. Again, we use a basic physics model for the electric motors. To obtain the power a motor is consuming, multiply the current torque the motor is providing in $N \cdot m$ by the rotational speed in rad/s. In our case the rotational speed is the wheel speed, ω_r or ω_l, since the motors are directly attached to the wheels. This value is the power drawn by the motor in $Watts$. To get the energy drawn, multiply the power by the time step over which the torque is applied. This model is described in the following equation.

$$MotorEnergyDraw(W \cdot h) = T(N \cdot m) \times \omega(rad/s) \times h(s) \tag{4.27}$$

Here h is the current time step and ω is the current wheel speed. The remaining components of the robotic vehicle that draw power will be modeled in a later version of this simulation. To implement the battery model in the UMBRA environment, a separate module is created to interface with the simulated servo module. This module has four input connectors representing the motors, cpu, camera, and pan and tilt unit power inputs respectively. The connectors for the cpu, camera, and pan and tilt unit are left open to be modeled later. These connectors are either connected to additional modules that model the power consumption behavior, or are set to a constant value using a Tcl set command. For example, to set the *cpu*

connector to 5 kJ enter `battery cpu_pwr 5` in the command window. Also for example, a module calculates the power consumed by a device based on its use and/or empirical data, such as an openGL camera taking video of the virtual world, and then passes the power draw value to the battery module through the connector.

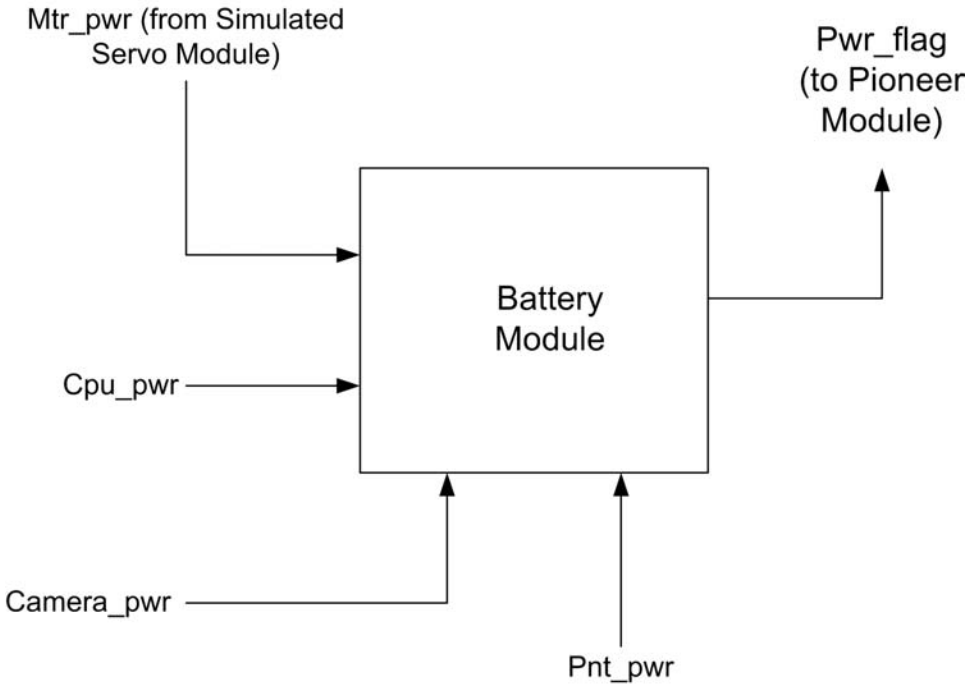


Figure 4.14: Diagram of the Battery Power Module

The motor power connector receives data from the simulated servo module, where the energy from the motors is calculated using equation 4.27. In the servo module, the motor power consumption is calculated in a separate function called `MotorPwr()` using the estimated wheel angular rates. Within the battery module, the energy draw from each of the devices is added together and then subtracted from the total energy from the battery. This process continues in real time as torque commands are given during the simulation. Once the battery

energy level reaches below a certain percentage of the total, there is no power available to the motors. To implement this effect, a power flag is sent back to the pioneer module which signals to the module to ignore any commands after the battery power is drained. A diagram of the battery module is included in Figure 4.14.

Battery Power Test Case

In order to test the battery module with the simulation, we initialize the battery with a low voltage and capacity so it drains prematurely. Next, the robotic vehicle simulation is activated and the cart is given different movement commands until the battery is drained and the cart cannot be controlled anymore. The battery power is then logged into a text file and plotted versus time. Figure 4.15 shows a plot of the power profile.

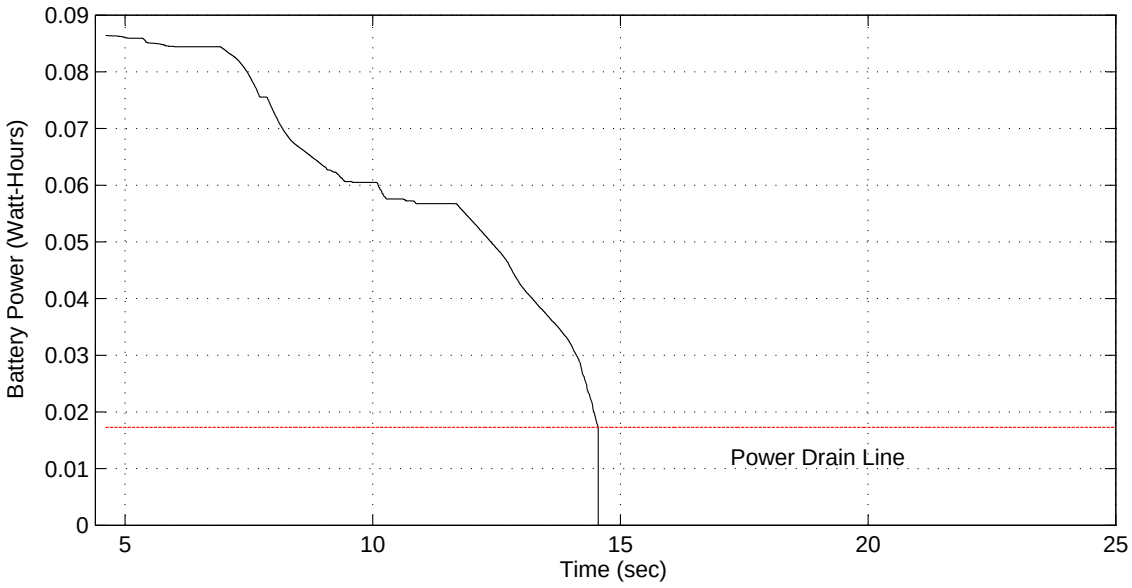


Figure 4.15: Battery Power vs. Time

In the figure a power drain line is drawn to represent 20 percent of the total charge, which is where the power drops to zero and the stop flag is sent to Pioneer. The battery capacity is set according to the number of batteries in use. The black line shows the power level dropping as the robotic vehicle controlled. Then, once the 20 percent level is reached, the power drops to zero and the vehicle is disabled. This basic model illustrates the behavior of a real vehicle when the battery runs out of power. In later versions, models can be written for the computer, camera, and pan and tilt unit and implemented in modules to be connected to this battery module. The battery capacity and voltage rating are set on the fly to represent one or more batteries. This battery module is not limited to this vehicle, it applies to any number of vehicles such as spacecraft or other robotic devices that run on batteries. This module represents the last component of the vehicle simulation. For reference, appendix tables 1, 2, and 4 list commands for the various modules and their purposes.

4.5 Orbit Propagator Module

The intent of this section is to implement the general orbit simulation of Chapter 3 into the UMBRA environment. This module creates a useful simulation tool to study both general and relative orbit problems. However, this tool is also designed to interface with other simulations which are discussed later.

4.5.1 Layout

In the orbit propagator module, there are three input connectors and two output connectors. The first connector is the `craft_flag` input connector. This connector passes a value containing the number of craft that are present in the simulation. This number can also be one, which corresponds to a single spacecraft orbit simulation. There is also a check against a maximum craft value. However the maximum number of craft can be changed by the user within the code. The next input connector is the `pert_flag` connector. This connector corresponds to the perturbations to be used in the equations of motion. For this connector, the user selects to study either an individual perturbation or all perturbations during the simulation. The different possible values of this connector are described in Table 4.4. Also included is Figure 4.16 describing the internal layout of the module. The last input connector is the `timestep` connector. This connector interfaces with either the UMBRA wallClock or simClock modules. This feature enables the simulation to have the option of running in simulated time or real time. The two output connectors `rel_motion` and `inert_motion`

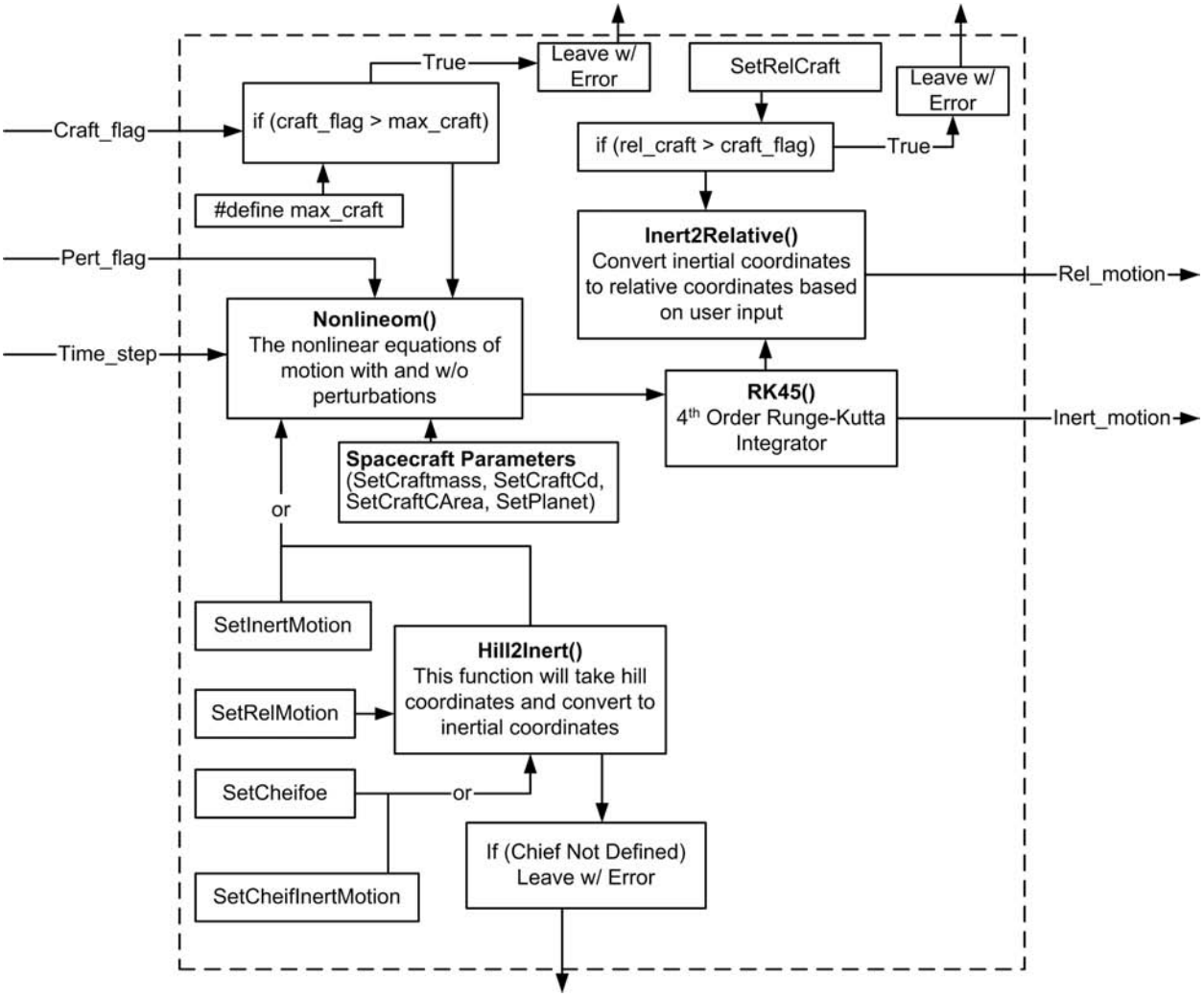


Figure 4.16: Diagram of the Orbit Module.

Table 4.4: Values for Perturbation Flag

Value	Description
0	No Perturbations
1	Gravitational Harmonics (J_2)
2	Atmospheric Drag
3	Solar Radiation Pressure
4	All Perturbations

output the relative motion and inertial motion of each of the spacecraft in question. If there is only a single spacecraft, then only the inertial motion would be of interest. In addition to these connectors, the user also specifies which craft to describe relative motion about through the `SetRelCraft` command. This command can have a value of 0, which corresponds to center of mass, or a value corresponding to a specific craft in the formation. The function value is also checked against the number of craft to ensure the user does not select a craft that does not exist. In this module, most of the algorithms and equations of motion have been discussed already and are implemented in various different functions. For example, the non-linear equations of motion are implemented in the `Nonlineom()` function within the main module. These functions and the various input checks are also featured in the diagram of Figure 4.16.

4.5.2 Integration

One new function is an integrator function that propagates these equations of motion through time. The integrator is implemented as a internal function to the module instead of an additional module. This feature allows for the module to have more control over the integration process. However, later iterations of this simulation may feature a universal integrator module. The integration technique used in the orbit module is a 4th-order Runge-Kutta method, which is described below.[6]

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) \tag{4.28}$$

$$\boldsymbol{k}_1 = h \cdot \boldsymbol{f}(\boldsymbol{x}) \quad \boldsymbol{k}_2 = h \cdot \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{k}_1/2)$$

$$\boldsymbol{k}_3 = h \cdot \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{k}_2/2) \quad \boldsymbol{k}_4 = h \cdot \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{k}_3)$$

Here $\boldsymbol{x}$ is the state and $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ represent the equations of motion. The reason for using this integrator is that while integrating the inertial motion, we are also computing the relative motion. The size of our formations is generally on the order of 10s of meters, while inertial motion is on the order of 1000s of kilometers. Therefore, small errors in the inertial motion translate into large relative motion errors. Another important note is that this integrator works for constant time step and variable time step cases. A higher order Runge-Kutta method could be considered to increase accuracy for large time step cases.

4.5.3 Initial Conditions

The last component to this simulation is initial conditions. The user is given different options to specify the initial positions and velocities of each spacecraft in the simulation. The user specifies either the inertial motion of each craft separately, or the relative motion to a chief orbit. If the latter choice is taken, there are additional options in selecting the chief orbit. The chief orbit is defined either as orbital elements or inertial position and velocity vectors. There are also checks in place in case the user attempts to specify relative motion without a chief orbit. Finally, Table 5 of the appendix describes the functions for setting parameters of the simulation. The next section details an interface module that links the vehicle and orbit simulations.

4.6 Relative Orbit Pioneer Interface Module

The last module to be created is called the Relative Orbit Pioneer Interface or ROPI for short. This module is an interface between the relative orbit simulator and the pioneer robotic vehicle simulation module. However, due to the design of the pioneer simulation, this interface works on the actual vehicle module as well. The overall result is that either the actual or simulated vehicle behaves like a spacecraft in orbit. This interface is a useful tool when studying relative motion concepts for spacecraft such as rendezvous and docking or formation keeping. A schematic that illustrates how this module enables this simulation concept is featured in Figure 4.17. In the figure, the orbit simulator integrates the equations

of motion for a spacecraft and passes the relative motion (position and velocity) to the interface module. The interface module generates the desired velocity commands using a feed-back control law that is designed to track the relative spacecraft orbit.

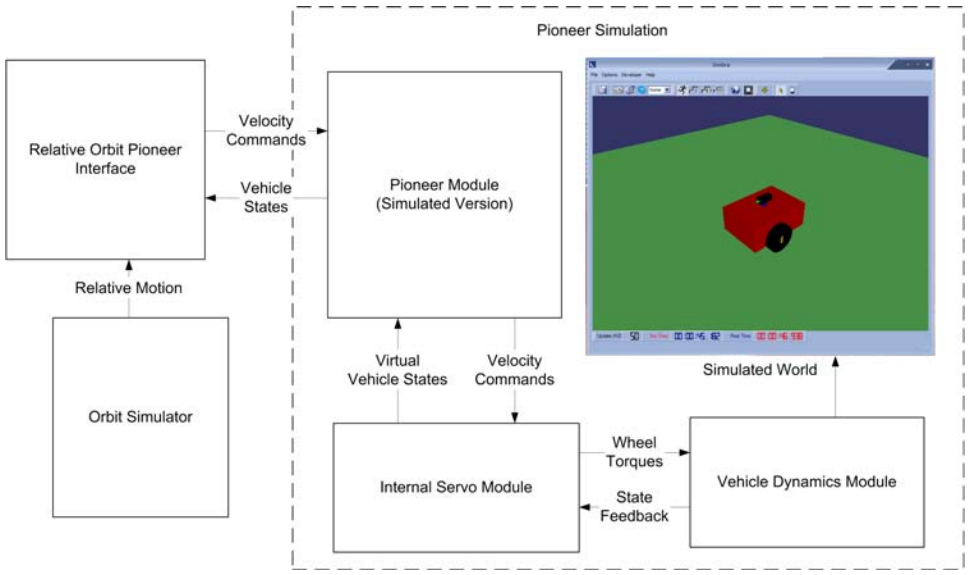


Figure 4.17: Schematic of Orbit/Pioneer Simulation

4.6.1 Layout

The ROPI module features two input connectors and one output connector. The first input connector, `pioneer_state`, receives the vehicles state from the pioneer module for feedback control. The second input connector, `rel_motion`, takes in the relative positions and velocities that are computed in the orbit simulator module. The sole output connector, `pmotion`, contains the velocity commands that are passed to the pioneer module in unguarded velocity mode. Its interface with the orbit simulator and pioneer is illustrated in Figure 4.18. The

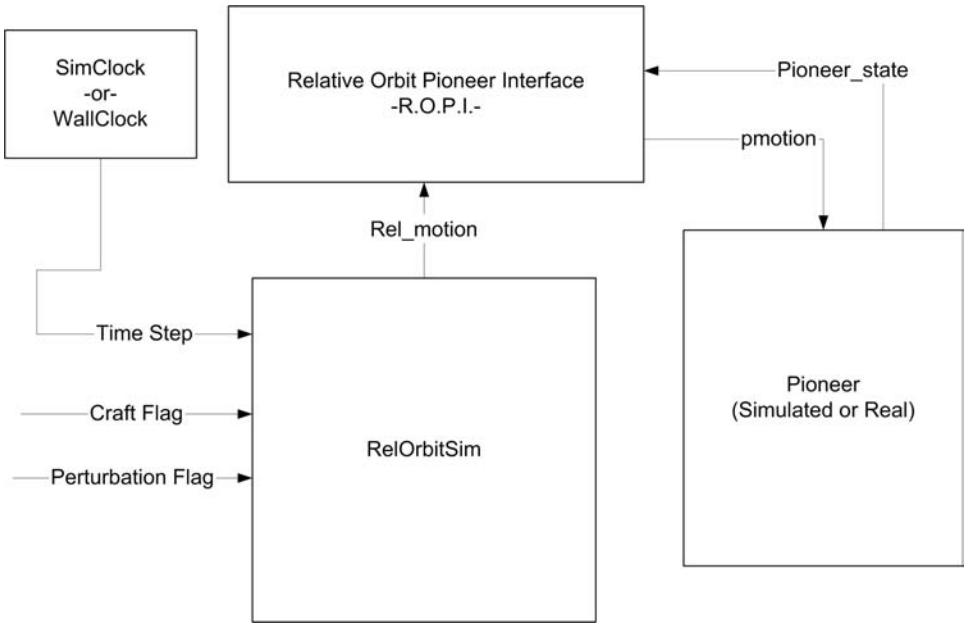


Figure 4.18: Diagram of ROPI Simulation.

next task is to define the simulation frame which represents the planar orbit tracking frame.

4.6.2 Simulation Frame

The goal of the ROPI module to make to robotic vehicle track the near-planar relative orbit motion of a spacecraft. An assumption is that the relative orbit of the spacecraft can be oriented in any way with respect to the chief orbit frame $\mathcal{H} : \{\hat{\boldsymbol{\imath}}_r, \hat{\boldsymbol{\imath}}_\theta, \hat{\boldsymbol{\imath}}_h\}$. Therefore, a simulation frame is defined such that the orbit plane aligns with the ground plane in which the vehicle moves. This frame is described in Figure 4.19 and the unit direction vectors

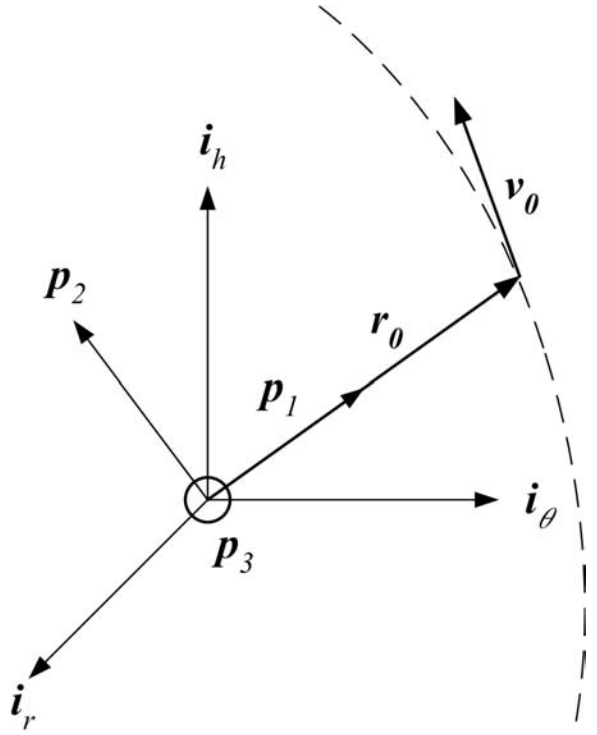


Figure 4.19: Simulation Frame $\mathcal{P}$

$\mathcal{P} : \{\hat{\boldsymbol{p}}_1, \hat{\boldsymbol{p}}_2, \hat{\boldsymbol{p}}_3\}$ are defined by:

$$\hat{\boldsymbol{p}}_1 = \frac{\boldsymbol{r}_0}{r_0} \qquad \hat{\boldsymbol{p}}_2 = \hat{\boldsymbol{p}}_3 \times \hat{\boldsymbol{p}}_1 \qquad \hat{\boldsymbol{p}}_3 = \frac{\boldsymbol{r}_0 \times \boldsymbol{v}_0}{|\boldsymbol{r}_0 \times \boldsymbol{v}_0|}$$

Here $\boldsymbol{r}_0$ and $\boldsymbol{v}_0$ are the initial position and velocity of the spacecraft in the chief orbit frame. It is important to note that this frame is defined initially and is held constant with respect to the rotating orbit frame. Finally, the rotation matrix $[PO]$, which maps vector components taken with respect to the orbit frame to the the simulation frame, is defined as

$$[PO] = \begin{bmatrix} \hat{\boldsymbol{p}}_1 & \hat{\boldsymbol{p}}_2 & \hat{\boldsymbol{p}}_3 \end{bmatrix}^T \tag{4.29}$$

The components of the position and velocity vectors of the spacecraft are represented in the simulation frame $\mathcal{P}$ as:

$$^{\mathcal{P}}\boldsymbol{r} = \begin{bmatrix} x & y & 0 \end{bmatrix}^T \quad ^{\mathcal{P}}\boldsymbol{v} = \begin{bmatrix} V_x & V_y & 0 \end{bmatrix}^T \tag{4.30}$$

The third component in these vectors is ignored since it is the out-of-plane component. In the simulation, it is assumed that this component is generally small.

4.6.3 Orbit Tracking Control Law

The sole function of this module is to control the vehicle through translational and rotational velocity commands and make it track the spacecraft's planar relative orbit. The spacecraft's and the vehicle's attitude are not considered, only the translational motion. Figure 4.20 is a description of the tracking problem with heading angle θ.

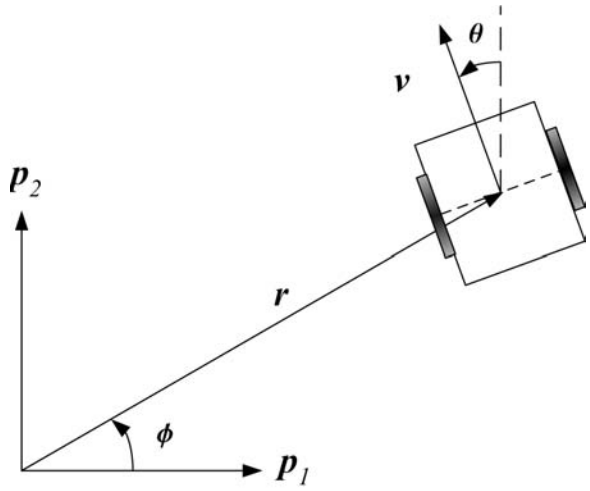


Figure 4.20: Tracking Problem

To design the control law, an existing potential function gradient-based control technique is used.[21] This control technique was originally designed to attract a two-wheeled vehicle to a stationary point by creating an attractive potential around a fixed target. Here this steering law is applied to the orbit tracking problem. Of interest is the performance of this control technique for a moving target. For this control law, the kinematic equations of motion for the vehicle are considered because the vehicle is controlled through speed commands. The kinematic equations of motion are repeated here for reference.

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = B(\boldsymbol{x})\boldsymbol{u} = \frac{1}{2} \begin{bmatrix} -R_r \sin\theta & -R_l \sin\theta \\ R_r \cos\theta & R_l \cos\theta \\ \frac{R_r}{L} & -\frac{R_l}{L} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \tag{4.31}$$

The control vector $\boldsymbol{u}$ has components ω_r and ω_l, which are the wheel speeds. First, a potential function on the state error is defined:

$$V = \frac{1}{2}\delta\boldsymbol{x}^T\delta\boldsymbol{x} \tag{4.32}$$

Here $\delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_d$ and $\boldsymbol{x}_d$ is the desired state vector. This potential function is positive definite with respect to the state error so that the vehicle states are attracted to the desired state. To obtain the control law, equation 4.32 is differentiated once to yield the potential rate function. This function is enforced to be negative definite with respect to the state error. Therefore the ideal control law drives the states to the desired values.

$$\dot{V} = \delta\boldsymbol{x}^T\delta\dot{\boldsymbol{x}} = -\delta\boldsymbol{x}^T[K]\delta\boldsymbol{x} \tag{4.33}$$

Here $[K]$ is a positive definite gain matrix. This potential rate function leads to the desired

stable closed-loop dynamics:

$$\delta\dot{\boldsymbol{x}} + [K]\delta\boldsymbol{x} = \dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d + [K]\delta\boldsymbol{x} = 0 \tag{4.34}$$

The next step is to substitute the kinematic equations of motion for $\dot{\boldsymbol{x}}$ and solve for the control vector $\boldsymbol{u}$:

$$\boldsymbol{u} = (B^T B)^{-1} B^T (\dot{\boldsymbol{x}}_d - [K]\delta\boldsymbol{x}) \tag{4.35}$$

Here, a least squares inverse is implemented on the matrix B since it is not a square matrix and therefore directly invertible. The consequence is that we cannot always achieve the desired closed-loop dynamics. This fact becomes clear when the potential rate function is computed later in this section. Also in this equation the state error vector $\delta\boldsymbol{x}$ and desired state derivative $\dot{\boldsymbol{x}}_d$ are defined as:

$$\delta\boldsymbol{x} = \begin{pmatrix} x - x_d \\ y - y_d \\ \theta - \theta_d \end{pmatrix} \quad \dot{\boldsymbol{x}}_d = \begin{pmatrix} V_x \\ V_y \\ \dot{\theta}_d \end{pmatrix} \tag{4.36}$$

Here x_d and y_d are the position coordinates of the spacecraft in the simulation frame. Additionally, the desired heading angle is defined such that the vehicle orients itself in the direction of the error.

$$\theta_d = \arctan\left(\frac{\delta x}{-\delta y}\right) = \arctan\left(\frac{x - x_d}{-(y - y_d)}\right) \tag{4.37}$$

This desired heading angle is differentiated once to yield the desired heading rate:

$$\dot{\theta}_d = \frac{-\delta y \delta\dot{x}}{(\delta x^2 + \delta y^2)} + \frac{\delta x \delta\dot{y}}{(\delta x^2 + \delta y^2)} \tag{4.38}$$

Finally, the components of the control vector $\boldsymbol{u}$ are converted to translational velocity and rotational velocity commands:

$$\dot{\theta}_c = \frac{(\omega_l - \omega_r)R}{2L} \quad V_c = (\omega_r + \omega_l)\frac{R}{2} \tag{4.39}$$

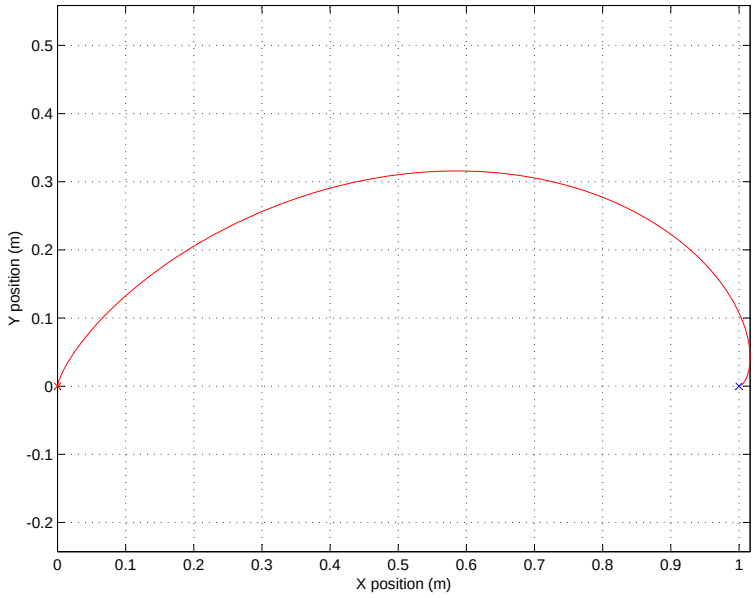
These commands are sent to the pioneer module via the `pmotion` connector. Also the gain matrix $[K]$ is defined as a diagonal matrix containing three positive gains for x, y, and heading angle θ. The gains are selected by the user through various Tcl functions in order to yield the desired closed-loop behavior. To analyze the stability of this control law, the control vector equation 4.35 is substituted back into the potential rate function in equation 4.33.

$$\dot{V} = \delta\boldsymbol{x}^T(B \cdot (B^T B)^{-1} B^T(\dot{\boldsymbol{x}}_d - [K]\delta\boldsymbol{x}) - \dot{\boldsymbol{x}}_d) \tag{4.40}$$

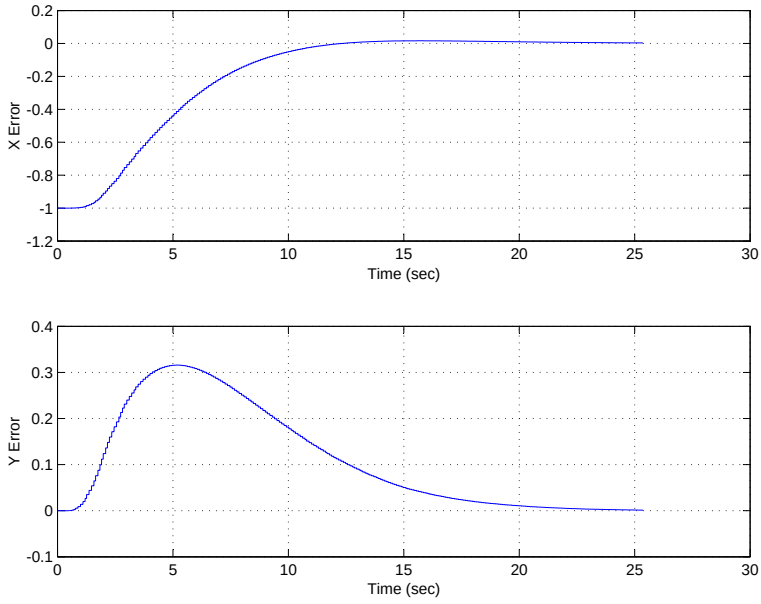
$$B \cdot (B^T B)^{-1} B^T = \begin{bmatrix} \sin^2\theta & -\cos\theta\sin\theta & 0 \\ -\cos\theta\sin\theta & \cos^2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.41}$$

A specific part of the potential rate equation has been analytically computed using the kinematic equations of motion and shown in equation 4.41. This equation illustrates a degenerative case where $\theta = 0$ or 180 degrees. In this case, any error in the x direction cannot be corrected for directly; the vehicle must first rotate toward this direction and then move forward. This also applies for error in the y direction when $\theta = 90$ or 270 degrees. Further analysis on this equation is necessary to prove that it is negative definite for all cases. However, a simple test case illustrates that this control law works on such a singularity. Here a stationary point is located at $x_d = 1$ m and $y_d = 0$ m, and the control law drives the vehicle

to this point. Also note that the vehicle's initial heading, θ_0, is zero. Figure 4.21 shows the $x - y$ position and error plots. From the figure, the vehicle converges on this point quickly and remains there. However, there is a small delay in the error convergence. This delay is due to the fact that the vehicle must first turn in place toward the desired point before moving forward. This behavior illustrates the degenerative case mentioned earlier. Now that the control law is able to move the vehicle to a stationary point, the next step is to track a moving point in an orbit. The following orbit parameters for this numerical simulation are featured in table 4.5. Additionally, the simulation is tracking one of two craft in a bounded elliptical relative orbit. Figure 4.22 displays the orbit tracking and the x and y error. The plots show that the vehicle tracks the orbit smoothly for one orbit period and the error remains bounded at about ± 0.1 m. However, there are wide bands in the error plots due to the vehicle following a sinusoidal path around the trajectory. In this simulation case the orbit interface runs at 1 Hz and the vehicle servo module updates at 2 Hz. Additionally, Figure 4.23 shows a the same test case with atmospheric drag turned on. The plots show similar tracking behavior as the spacecraft spiral away from each other. The simulation test case illustrates only a first iteration for an orbit tracking control law. Improvements on this control law include refining the gain selection techniques and finding the best gains for relative orbit tracking. In this case, the gains were selected by trail and error in order to get the smoothest possible tracking behavior. However, a new control law may need to be developed in order to achieve better performance.

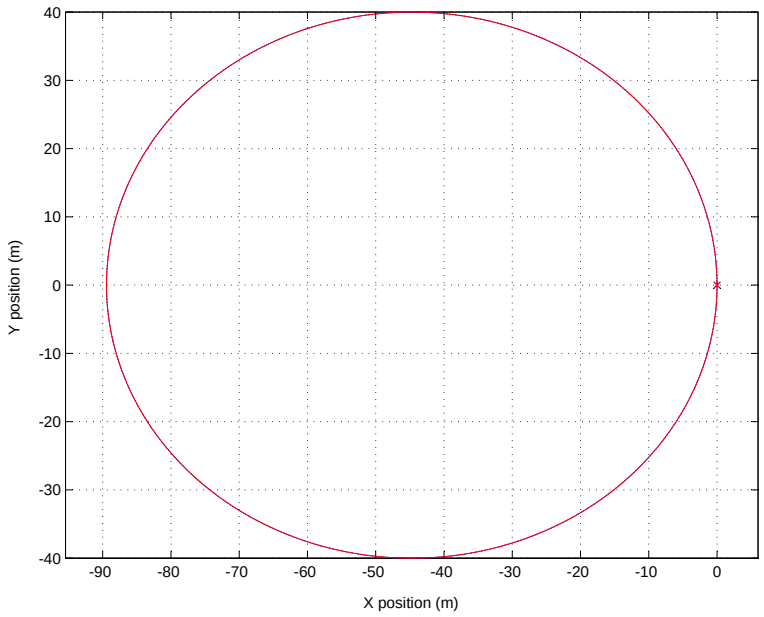


(a) $x - y$ Position Plot

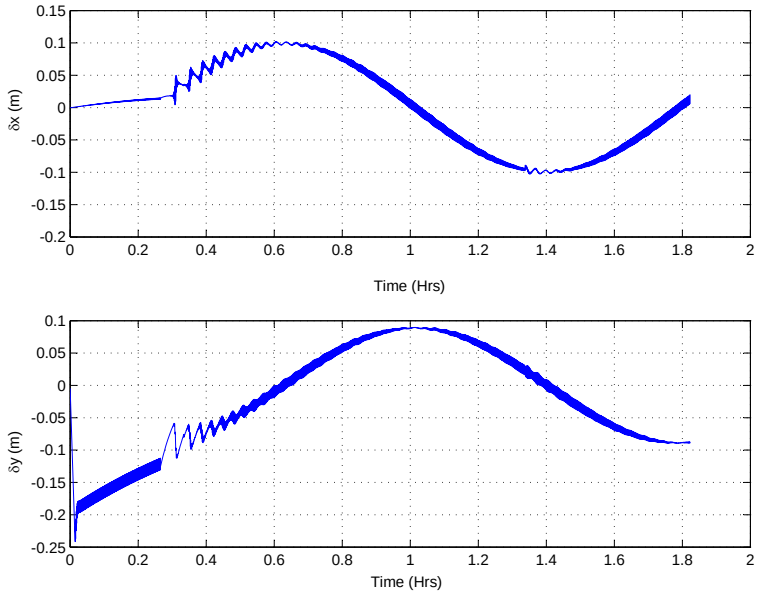


(b) $x - y$ Position Error

Figure 4.21: Stationary point tracking with $k_x = k_y = k_\theta = 0.3$

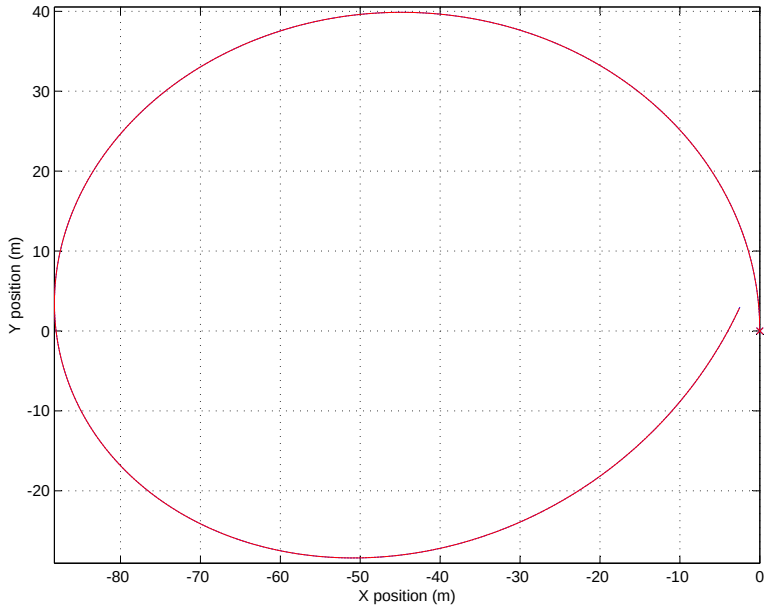


(a) $x - y$ Position Plot

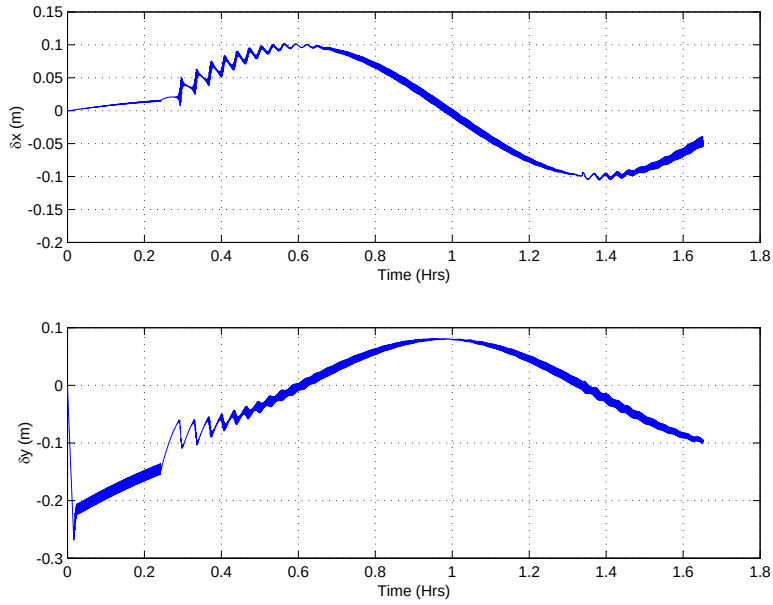


(b) $x - y$ Position Error

Figure 4.22: Spacecraft relative orbit tracking with $k_x = k_y = 0.005, k_\theta = 0.05$



(a) $x - y$ Position Plot



(b) $x - y$ Position Error

Figure 4.23: Spacecraft relative orbit tracking with Atmospheric Drag and $k_x = k_y = 0.005, k_\theta = 0.05$

Table 4.5: Initial Chief (Center of Mass) Orbit Parameters

Parameter	Value
Semi-major Axis	6800 km
Eccentricity	0.0
Inclination	0.7854 rad
Right Ascension of Ascending Node	0.3491 rad
Argument of Periapsis	0.2618 rad
Anomaly Angle	0.0 rad
C_{d1} and C_{d2}	2.6 or 2.0
A_1 and A_2	0.7854 or 1.5000 m^2

Chapter 5

Conclusion and Future Work

The relative motion problem is both an interesting and complex concept to study. In this thesis, simulation tools are developed to study different aspects of this problem. Specifically, a numerical simulation is developed to mimic a 2-wheel driven unmanned vehicle which includes a friction model to simulate slippage on a variety of different environments. Also, a relative orbit simulator is developed using the full non-linear equations of motion of a spacecraft. The orbit simulation includes various perturbation models to study their effects on the relative motion of spacecraft. These perturbation models include gravitational harmonics, atmospheric drag, and solar radiation pressure. In addition to developing these simulations analytically, they are also implemented within the UMBRA framework. These simulation modules interact with each other or other modules and are manipulated in real time. In addition, an interface module is created to link the orbit and vehicle simulations, thus allowing the simulated or actual vehicle to track a spacecraft's relative orbit.

Although this problem has been studied in a variety of different lab environments, the AVS Lab provides a unique perspective on the study of relative motion concepts. Further research to be done in this lab includes developing simulations for other types of vehicles as well as simulating new sensing techniques. The visual sensing techniques that are currently being explored in hardware are also being developed in simulation. There are also some immediate improvements that can be done on the research presented in this thesis. For instance, a more robust friction model needs to be developed to include such concepts as rolling friction and kinetic friction. Another improvement is the graphical representation of the autonomous vehicle and its simulated environment. With current UMBRA software, it is possible to import textures and other three-dimensional models. These improvements allow the virtual vehicle to explore more realistic environments in the simulated world. Additionally, more accurate vehicle parameters such as weight and moments of inertia need to be determined.

Bibliography

[1] Aerospace and Virginia Tech Ocean Engineering Department. Space systems simulation laboratory. http://www.sssl.aoe.vt.edu. Blacksburg, VA.

[2] Kyle T. Alfriend and Hanspeter Schaub. Dynamics and control of spacecraft formations: Challenges and some solutions. *Journal of the Astronautical Sciences*, 48(2 and 3):249–267, April–Sept. 2000.

[3] Kyle T. Alfriend, Hanspeter Schaub, and Dong-Woo Gim. Gravitational perturbations, nonlinearity and circular orbit assumption effect on formation flying control strategies. In *Proceedings of the Annual AAS Rocky Mountain Conference*, pages 139–158, Breckenridge, CO, Feb. 2–6 2000.

[4] Pond B. and I. Sharf. Experimental demonstrator of flexible manipulator trajectory optimization. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 1869–1876, Reston, VA, USA, 1999. AIAA.

[5] Richard H. Battin. *An Introduction to the Mathematics and Methods of Astrodynamics*. AIAA Education Series, New York, 1987.

[6] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Brooks/Cole, 2001.

[7] Rich Burns, Craig A. McLaughlin, Jesse Leitner, and Maurice Martin. Techsat 21: formation design, control, and simulation. In *IEEE Aerospace Conference Proceedings*, volume 7, pages 19–25, Big Sky, MO, March 18–25 2000.

[8] H. Schaub G. G. Parker C. C. Romanelli, A. Natarajan and L. B. King. Coulomb spacecraft voltage study due to differential orbital perturbations. In *AAS Space Flight Mechanics Meeting*, number AAS 06-123, Tampa, FL, January 22-26 2006.

[9] W. H. Clohessy and R. S. Wiltshire. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences*, 27(9):653–658, Sept. 1960.

[10] H. Fornoff. Final report for air bearing platform t50-2. Technical Report CR-97588, NASA, October 1967.

[11] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, Menlo Park, CA, 1998.

[12] K. E. Glover. Development of a large support surface for an air-bearing type zero-gravity simulator. Technical Report TM-X-72780, NASA, April 1976.

[13] George William Hill. Researches in the lunar theory. *American Journal of Mathematics*, 1(1):5–26, 1878.

[14] Vikram Kapila, Andrew G. Sparks, James M. Buffington, and Qiguo Yan. Spacecraft formation flying: Dynamics and control. In *Proceedings of the American Control Conference*, pages 4137–4141, San Diego, California, June 1999.

[15] S. A. Kowalchuk and C. D. Hall. Distributed spacecraft attitude control system simulator feedback control capabilities and visualization techniques. In *7th International Symposium on Quantitative Feedback Theory (QFT) and Robust Frequency Domain Design Methods*, University of Kansas, August.

[16] Yoshihara K. Takahashi T. Tsurumi S. Matunaga, S. and K. Ui. Ground experiment system for dual-manipulator-based capture of damaged satellites. In *Proceedings of the IEEE International Conference on Intelligent Robotic Systems*, pages 1847–1852, Piscataway, NJ, USA, 2000. IEEE.

[17] University of Florida Mechanical Engineering Department. Nonlinear controls and robotics group. http://www.mae.ufl.edu/ dixon/facilities.htm. Gainsville, FL.

[18] U.S. Government Printing Office. U.s. standard atmosphere. Washington, D.C., 1976.

[19] Archie E. Roy. *Orbital Motion*. Adam Hilger Ltd, Bristol, England, 2nd edition, 1982.

[20] Hanspeter Schaub and Kyle T. Alfriend. j_2 invariant relative orbits for spacecraft formations. *Celestial Mechanics and Dynamical Astronomy*, 79(2):77–95, 2001.

[21] Hanspeter Schaub and John L. Junkins. Dynamics and control of micro-robot swarms: Planar motion and state estimation. Technical report, Texas A&M University.

[22] Hanspeter Schaub and John L. Junkins. *Analytical Mechanics of Space Systems*. AIAA Education Series, Reston, VA, October 2003.

[23] H. Schuber and J. How. Space construction: An experimental testbed to develop enabling technologies. In *Proceedings of the Conference on Telemanipulator and Telepresence Technologies IV*, pages 179–188, Piscataway, NJ, USA, 1997. IEEE.

[24] Jana L. Schwartz, Mason A. Peck, and Christopher D. Hall. Historical review of air-bearing spacecraft simulators. *AIAA Journal of Guidance, Control and Dynamics*, 26(4):513–522, 2003.

[25] Raymon Sedwick, David Miller, and Edmund Kong. Mitigation of differential perturbations in clusters of formation flying satellites. In *AAS/AIAA Space Flight Mechanics Meeting*, February 1999. Paper No. AAS 99-124.

[26] Prasenjit Sengupta and Srinivas R. Vadali. A lyapunov-based controller for satellite formation reconfiguration in the presence of j_2 perturbations. In *AAS Space Flight Mechanics Meeting*, Maui, Hawaii, February 8–12 2004. Paper No. AAS-04-253.

[27] K. Toshihiro. Solar radiation pressure model for the relay satellite of selene. *Earth Space and Planets*, 51:979–986, September 1999.

[28] D. M. Dawson W. E. Dixon, M. S. de Queiroz and T. J. Flynn. Adaptive tracking and regulation control of a wheeled mobile robot with controller/update law modularity. *IEEE Transactions on Control Systems Technology*, 12(1):138–147, 2004.

[29] James R. Wertz and Wiley J. Larson. *Space Mission Analysis and Design.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

Appendix

This appendix contains tables that describe the Tcl/Tk functions for setting parameters in the vehicle and relative orbit simulations. It is organized by each module that is a part of these simulations.

Table 1: Cart Dynamics Module Commands

Command Name	Description	Default Value
`SetGravity`	Set the value of gravity	9.81
`SetMun`	Lateral friction coefficient	2
`SetMufl`	Forward friction coefficient for left wheel	0.24
`SetMufr`	Forward friction coefficient for right wheel	0.24

Table 2: Battery Power Module Commands

Command Name	Description	Default Value
`setBattVoltage`	Total voltage of the battery(ies)	0.12
`setBattCap`	Total capacity of the battery(ies)	0.72

Table 3: ROPI module Commands

Command Name	Description	Default Value
`SetXGain`	Set the gain on X position	0.1
`SetHeadGain`	Set the gain on heading angle	0.1
`SetYGain`	Set the gain on Y position	0.1
`SetCraft`	Set which craft to track	2

Table 4: Simulated Servo Module Commands

Command Name	Description	Default Value
`SetWheelRadius`	Size of the Wheel (R) in m	0.098
`SetWheelBase`	Distance between the two wheels (L) in m	0.33
`SetWheelInertia`	Inertia of the wheel (I_{w1}) in kgm^2	0.0025
`SetWheelMass`	Mass of the wheel (m_w) in kg	0.5
`SetCartMass`	Mass of the vehicle body (m_c) in kg	8
`SetCartInertia`	Inertia of the vehicle body (I_{c3}) in kgm^2	0.21
`SetWheelInertia3`	Inertia of the wheel (I_{w3}) in kgm^2	0.001305
`SetFreeWheeldist`	Distance of the support wheel (d_1) in m	0.225
`SetAxisdist`	Distance of the wheel axis (d_2) in m	0.0
`SetSpeedGain`	Wheel speed gain on torque inputs	0.03
`SetPosGain`	Wheel position gain on torque inputs	0.0
`SetRunFreq`	The run frequency of the servo module in Hz	10
`SetLPFreq`	The low pass filter cut-off frequency in Hz	0.5
`SetEncFreq`	The encoder filter cut-off frequency in Hz	5.0

Table 5: RelOrbitSim module Commands

Command Name	Description	Default Value
`SetChiefoe`	Set Chief Orbital Elements	User Defined
`SetRelMotion`	Relative position and velocities	User Defined
`SetInertMotion`	Inertial position and velocities	User Defined
`SetCheifInertMotion`	Inertial position and velocity of the cheif	User Defined
`SetCraftmass`	The mass in kg	{50 ...}
`SetCraftCd`	The coefficient of drag	{0 ...}
`SetCraftCArea`	The cross-sectional area in m^2	{0 ...}
`SetNCraft`	The number of spacecraft	2
`SetPert`	The perturbation flag	0
`SetCraftSunvec`	The sun vector of the spacecraft in AU	{0 0 1}
`SetPlanet`	The planet the spacecraft are orbiting	3(Earth)

Vita

Christopher Craig Romanelli was born on August 21st, 1982 in Warwick, Rhode Island. He graduated from Western Branch High School in Chesapeake, Virginia. Chris began his college career at Virginia Tech in the Fall of 2000 and graduated with a Bachelor's Degree in Aerospace Engineering in the Spring of 2004.