Real-time Minimum Jerk Optimal Trajectory Synthesis and Tracking for Ground Vehicle Applications

by

NAGA VENKATA MURALI VEERAPANENI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2017

To my mom and dad for giving me this opportunity and believing in me.

ACKNOWLEDGEMENTS

First of all I want to thank my supervisor Dr. Kamesh Subbarao, for believing in me and giving me this wonderful opportunity to learn. I would like to thank him for guiding me and teaching me all those interesting and useful stuff which made me tackle the problem in a practical sense.

I would like to thank my committee members Dr. Seiichi Nomura and Dr. Panayiotis Shiakolas for taking time to serve on the committee. Dr. Shiakolas taught me classical control theory which got me fascinated about the control theory and got me started as a controls researcher.

I would like to thank my friends at the Aerospace Systems Laboratory, Pengkai, Ameya, Denish, Paul, Kelvin, Roopak, Karan, Rajnish, Madhu, Tracie, Alok, Ziad, Pavan, Diganta and Abel for making it fun during my master's program. To Pengkai and Denish for helping me when there is an emergency. Special thanks to Pengkai for making me learn the programming, helping me during interviews and helping me in tough times.

Special thanks to my Father and Mother for believing in me and giving me this opportunity and all the sacrifices that they made. This degree is as much as my father's as it is mine. Thanks to my sister and my brother-in-law for all the fun and support.

I would like to thank Air Force Research Laboratory (AFRL), for providing funding for the research via AFRL Award # FA9453-16-1-0058.

November 17, 2017

ABSTRACT

Real-time Minimum Jerk Optimal Trajectory Synthesis and Tracking for Ground Vehicle Applications

NAGA VENKATA MURALI VEERAPANENI, M.S

The University of Texas at Arlington, 2017

Supervising Professor: Kamesh Subbarao

Research on wheeled mobile robots has been an active research field for several decades with focus on the problems of stability, maneuverability and control. The trajectory generation for wheeled mobile robots is usually handled by considering a smooth function satisfying the boundary conditions. Even though this problem is well posed for several function classes, the real issue occurs in generating a feasible trajectory for the robot which takes into account the constraints of the system. Another problem from the control system perspective is, whether the derived controller is stable for the entire trajectory span to track the given trajectory.

The purpose of this thesis is to provide a different approach for trajectory generation and control of an autonomous ground vehicle. The presented trajectory generation technique takes into account the acceleration constraint of the system by performing an optimization routine in-order to obtain the final time of the trajectory for a given waypoint. Then, the problem of tracking the desired trajectory is handled by using a nonlinear backstepping control law which takes into account the non-holonomic constraints of the robot. The overall experimental setup is based on

the cyber-physical system architecture by separating the rover and the ground station, which handles all the necessary computation. The time delays associated with this kind of system architecture is characterized and presented. The presented trajectory generation and control techniques are experimentally verified by using the cyber-physical system architecture on a real mobile robot equipped with GPS, IMU, Wheel Encoder and LiDAR sensors.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Autonomous mobile robots have attracted researchers from different fields over the past years. Various path planning and control algorithms have been developed over these years, which makes the robot to go from it's initial configuration to the final desired configuration to do a specific task. Two main challenges in mobile robot are path planning and control. Path planning addresses the challenge of making the robot aware of it's environment and changing it's path in real-time, to account for the environmental constraints. One way to tackle this problem is to use a global path planner which takes the initial and final GPS co-ordinates and generates a set of way points for the robot to go through. The global path planner requires information about it's environment topology. Another way to solve the path-planning problem is to use a local path planner. The local path planner discretizes the target configuration into a subset of intermediate configurations and plans for a successive subset of configurations. The local planner is more reactive in that it can handle dynamic obstacles along it's path. Once the path is computed, a trajectory needs to be generated. If a mobile robot is required to reach the target position in a specified amount of time, then the designed path is dependent on time. Even though the problem of trajectory generation is well defined, the generated trajectory is required to satisfy the vehicle constraints along it's path like maximum acceleration, minimum-turning radius, non-holonomic constraints of the robot, etc.

Another challenge in making an autonomous mobile robot is the stability of

the robot over it's entire mission span and the ability to track the reference path or trajectory as close as possible. As the kinematics of the robot is nonlinear, one might think to linearize the system around it's equilibrium and design a linear controller. The drawback of this technique is, some good nonlinearities were neglected or some hard nonlinearities were not addressed which would eventually destablitize the system if operated beyond the robot's equilibrium point. So, a lot of research is focused on developing different nonlinear control laws which takes into account the nonlinearities of the system.

1.2 Related Work

There is an abundance of trajectory generation algorithms which takes into account the robot dynamical constraints. The main idea of trajectory generation is to find a necessary polynomial that satisfies pre-specified constraints. Fundamental methods of generating a suitable polynomial can be found in [1]. Most of the paths are generated by stitching together straight lines and circular arc segments, which is obtained by minimizing the path length. Dubins [2], Reeds-shepp [3] showed that the optimal path between two configurations can be obtained by a series of straight lines and circular arcs. Even though the dubins path is computationally straight forward, the main drawback is that the curvature is not continuous when the circle and line are linked to each other. To handle this drawback a clothoid curve or Cornu or Euler spiral is used to replace the circular arcs in dubins path. Clothoid is a curve with a continuous curvature, which varies linearly over the path. The clothoid path was used as a continuous cyurvature trajectory generation for UAVs[4]. Riesenfeld and Gordon used B-spline to generate a continuous trajectory [5]. The B-spline is a set of basis functions that has some properties such as affine invariance, local modification scheme, convex hull, continuity and differentiability. Because of this local modifica-

tion property B-spline based trajectory generation is more useful for online obstacle avoidance while maintaining continuity in trajectory. A convolution-based trajectory generation method has been suggested in [6, 7] which does not use any polynomial. The convolution-based method is able to generate an S-curve trajectory within the allowable physical system limits by applying successive convolution operations. The main advantage of the convolution based trajectory is, it can be used to generate the trajectory that satisfies the differential constraints of n^{th} order.

There has also been implementations of nonlinear control designs in a wide variety of mobile robot applications [8, 9, 10, 11, 12, 13, 14, 15]. Some of the applications of a nonlinear control to a wheeled robot have only been verified through simulation, as in [8, 9, 10, 11, 12], while others have been validated by hardware experiments such as references [13, 14, 15]. Dixon [8] provided a control design based on a virtual structure approach to follow a simple user defined trajectory. And in [9], an adaptive control design is implemented in simulation to control a two wheel like mobile robot. The adaptive control design is proven to be robust to input saturation and disturbances [9]. Backstepping control designs for wheeled mobile robots or rovers are discussed in [10, 11, 12]. The control laws in [10] and [11] apply backstepping to control the kinematic model directly from the dynamics of the system and torques on the vehicle are applied as the control inputs. In contrast, the backstepping control design in [12] accounts for regulation of the rover's heading angle turn rate and its forward acceleration.

1.3 Problem Description

This thesis addresses the problem of trajectory generation by considering the acceleration constraints of the robot to obtain a closed-form analytical solution and at the same time a new computationally straightforward nonlinear control law is tested

to make sure, that the proposed control law provides stability over the entire mission span. This thesis also validates the use of cyber-physical system architecture for further research on time-delay control approaches.

1.4 Thesis Outline

This thesis is organized as follows: In Chapter 2, the kinematic model of the mobile robot is derived and the controllability of the robot is analyzed. Chapter 3 details the design of Minimum-Jerk trajectory design for constrained acceleration trajectory. Chapter 4, introduces the nonlinear backstepping control and the modified version that is used to control the mobile robot. In Chapter 5, the experimental setup for testing the trajectory and control algorithms is presented. Chapter 6 and 7 presents the time-delay analysis and the experimental results. Finally, in Chapter 8, concluding remarks are stated.

CHAPTER 2

Modeling and Analysis of the Differential-Drive Robot

2.1 Kinematic Modeling

Kinematic modeling is the study of the motion of mechanical systems without considering the forces that affects the motion. For the differential-drive mobile robot, the main objective of the kinematic modeling is to represent the velocities of the robot as a function of the driving velocities of the wheels by considering the geometric parameters of the robot.

2.1.1 Coordinate Systems

To determine the position of the mobile robot in its environment, two different coordinates systems (or) frames need to be defined.

- Inertial Coordinate System: This coordinate frame is fixed in the environment or the plane in which the mobile robot traverses. It is denoted by $\{X_I, Y_I, Z_I\}$
- Body Coordinate System: This coordinate frame is fixed to the robot and it moves along with the robot. This frame denoted by $\{X_b, Y_b, Z_b\}$.

For the derivation of the kinematic model, the Inertial and body frame are assigned as shown in Fig. 2.1,

- Inertial frame is attached with X_I pointing towards local East, Y_I pointing towards local North, Z_I going up.
- Body frame is attached with X_b pointing robots forward direction, Y_b pointing towards robots left side, Z_b going up.

The heading angle, ψ, is positive, counter-clockwise direction about Z_I axis.

The transformation between the Inertial frame and the body frame is given as,

$$X^I = R(\psi)X^b \tag{2.1}$$

where

$$R(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This transformation can be used to handle motion between the frames,

$$\dot{X}^I = R(\psi)\dot{X}^b \tag{2.2}$$

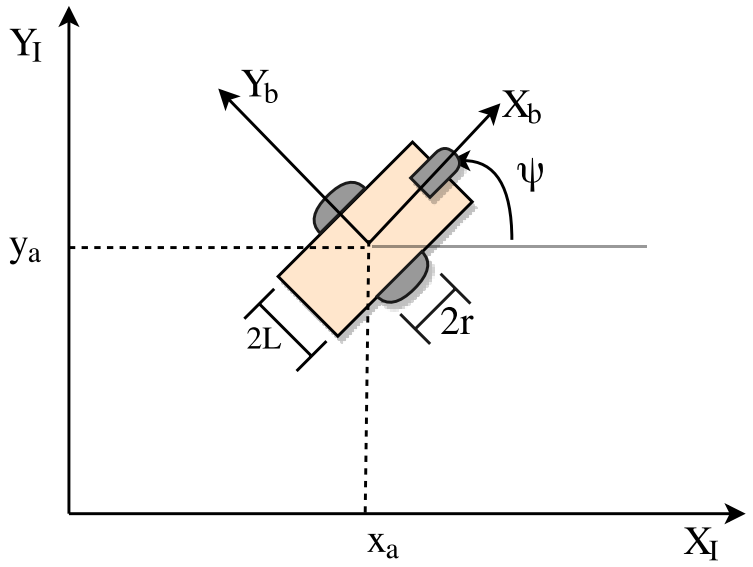


Figure 2.1: Inertial (I) and Body (b) coordinate frames

Let the position of the rover in the inertial frame be denoted by $\{x_a, y_a\}$ and in the body frame by $\{x_b, y_b\}$

2.1.2 Motion Constraints

The robot motion is characterized by two non-holonomic constraints. These non-holonomic constraints are obtained by assuming,

- No lateral motion: This constraint implies that the mobile robot can only go forward or backward but not sideways. This simply means that the velocity at the center of the vehicle along its lateral axis is zero,

$$\dot{y}_b = 0$$

By using the Eq. 2.2, the velocity in the inertial frame is obtained as,

$$-\dot{x}_a \sin\psi \; + \; \dot{y}_a \cos\psi = 0 \tag{2.3}$$

- Pure rolling motion: The pure rolling constraint assumes that each wheel maintains a point contact with the ground, without wheel slippage in the longitudinal axis (X_b) and no skidding in lateral axis (Y_b). The linear velocities of the wheels in the body frame $\{b\}$ is given as,

$$\begin{aligned} V_l &= r\omega_l \\ V_r &= r\omega_r \end{aligned} \tag{2.4}$$

where $V_l,\ V_r$ are the left and right wheel forward velocity, $\omega_l,\ \omega_r$ are the left and right wheel angular velocity, r, is the radius of the wheel. These velocities can be expressed in inertial frame as a function of robots state $\{x_a,\ y_a\}$ as,

$$\begin{aligned} \dot{x}_r &= \dot{x}_a + L\dot{\psi}\cos\psi \\ \dot{y}_r &= \dot{y}_a + L\dot{\psi}\sin\psi \end{aligned} \tag{2.5}$$

$$\begin{aligned} \dot{x}_l &= \dot{x}_a + L\dot{\psi}\cos\psi \\ \dot{y}_l &= \dot{y}_a + L\dot{\psi}\sin\psi \end{aligned} \tag{2.6}$$

using the transformation matrix $R(\psi)$, the rolling constraints in body frame is given as,

$$\begin{aligned} \dot{x}_l \cos\psi \ + \ \dot{y}_l \sin\psi = r\omega_l \\ \dot{x}_r \cos\psi \ + \ \dot{y}_r \sin\psi = r\omega_r \end{aligned} \tag{2.7}$$

By combining Eq. 2.3 & 2.7, the final motion constraints of the robot is written as,

$$\Lambda(q)\dot{q} = 0 \tag{2.8}$$

where

$$\Lambda(q) = \begin{bmatrix} -\sin\psi & \cos\psi & 0 & 0 & 0 \\ \cos\psi & \sin\psi & L & -r & 0 \\ \cos\psi & \sin\psi & -L & 0 & r \end{bmatrix} \tag{2.9}$$

and

$$q = \begin{bmatrix} \dot{x}_a & \dot{y}_a & \dot{\psi} & \omega_l & \omega_r \end{bmatrix} \tag{2.10}$$

Eq. 2.8 is characterized as a non-integrable constraint or non-holonomic. If the equation can be integrated, one can eliminate coordinates by using the equations of constraints as is in the case of holonomic constraints. As is seen, the systems containing non-holonomic constraints always require more coordinates for the description of state than there are degrees of freedom.

2.1.3 Rover Kinematics

As the linear velocity of each driving wheel of the robot, is the linear velocity of the robot itself, the linear velocity of the robot is given as,

$$V = \frac{r(\omega_l \ + \ \omega_r)}{2} \tag{2.11}$$

Due to the absence of the lateral motion, the robot can only move forward and not sideways, the forward and lateral velocity is given as,

$$\begin{aligned} \dot{x}_b &= \frac{r(\omega_l \ + \ \omega_r)}{2} \\ \dot{y}_b &= 0 \end{aligned} \tag{2.12}$$

The change in wheel angular velocity is the change in the angular rate of robot $\dot{\psi}$,

$$\dot{\psi} = \frac{r(\omega_r \ - \ \omega_l)}{2L} \tag{2.13}$$

By using the transformation matrix $R(\psi)$, the body velocities are expressed in inertial frame velocities as,

$$\begin{aligned} \dot{x}_a &= \frac{r(\omega_l \ + \ \omega_r)}{2} \cos\psi \\ \dot{y}_a &= \frac{r(\omega_l \ + \ \omega_r)}{2} \sin\psi \\ \dot{\psi} &= \frac{r(\omega_r \ - \ \omega_l)}{2L} \end{aligned} \tag{2.14}$$

Eqs. 2.14 are the robot velocities expressed in inertial frame and can be used for the stability, controllability analysis.

2.2 Controllability Analysis

Before going on to discuss about planning and control, first we need to make sure that the system is controllable. A system is said to be controllable, if the rank of the controllability matrix (C) is same as the number of states of the system i.e, the controllability matrix is of full rank. For a system of the form,

$$\dot{X} = AX + BU \tag{2.15}$$

the controllability matrix (C) is defined as,

$$C = \begin{bmatrix} B & AB & A^2B & A^3B & ... & A^{n-1}B \end{bmatrix} \tag{2.16}$$

where n is the order of the system.

Eq.2.14 can be written as,

$$\dot{q} = G_1(q)\omega_r + G_2(q)\omega_l \tag{2.17}$$

with $G_1 = \begin{bmatrix} \frac{r}{2}\cos\psi \\ \frac{r}{2}\sin\psi \\ \frac{r}{b} \end{bmatrix}$, $G_2 = \begin{bmatrix} \frac{r}{2}\cos\psi \\ \frac{r}{2}\sin\psi \\ -\frac{r}{b} \end{bmatrix}$

and q, is the generalized coordinates with $q = \begin{bmatrix} \dot{x}_a, & \dot{y}_a, & \dot{\psi} \end{bmatrix}^T$.

The kinematics Eq. 2.14 are nonlinear, driftless and have less control inputs than the number of generalized coordinates. A driftless system is a system for which no motion takes place under zero input. The following sections shows the analysis whether the system can be controllable about a stationary position and also along a trajectory.

2.2.1 Controllability at a point

As the system is driftless, any point corresponds to an equilibrium point provided that the input is zero.

let q_e be the equilibrium point, then the Eq. 2.17 becomes,

$$\Delta q = G_1(q_e)\omega_r + G_2(q_e)\omega_l = G(q_e)\left[\omega_l, \ \omega_r\right]^T \tag{2.18}$$

where $\Delta q = q - q_e$. From the above equation, we can write $A = 0, \ B = G(q_e)$. By using A, B, the rank of the controllability gramian is 2 from Eq. 2.19. Hence the linearized system is not controllable and linear controller won't work, not even locally [16].

$$C = \begin{bmatrix} B & AB & A^2B \end{bmatrix} = rank(2) \tag{2.19}$$

One way to truly look at the controllability of a driftless nonlinear system is to evaluate a Lie Algebra rank Condition [17] (nonlinear controllability)

$$rank\begin{bmatrix} G_1 & G_2 & [G_1,\ G_2] & [G_1,\ [G_1,\ G_2]] & \cdots \end{bmatrix} \tag{2.20}$$

where $[G_1,\ G_2]$ is Lie bracket and is evaluated as,

$$[G_1,\ G_2] = \frac{\partial G_2}{\partial q}\ G_1\ -\ \frac{\partial G_1}{\partial q}\ G_2 \tag{2.21}$$

By using the Lie Algebra Rank Condition(LARC), the rank of the system is 3 from Eq. 2.22. This means that the robot is controllable. This also proves that linear controllability is not a reliable indicator for nonlinear systems.

$$LARC = [G_1\ G_2\ [G_1,\ G_2]] = \begin{bmatrix} \frac{r}{2}\cos\psi & \frac{r}{2}\cos\psi & -\frac{r^2}{b}\sin\psi \\ \frac{r}{2}\sin\psi & \frac{r}{2}\sin\psi & \frac{r^2}{b}\cos\psi \\ \frac{r}{b} & -\frac{r}{b} & 0 \end{bmatrix} = rank(3) \tag{2.22}$$

2.2.2 Controllability about a trajectory

Let the desired trajectory for the states be $q_d(t) = \{x_d(t), y_d(t), \psi_d(t)\}$ and the reference control be $\omega_d(t) = \{\omega_{dl}(t), \omega_{dr}(t)\}$. Linearizing Eq. 2.17 about this reference trajectory gives,

$$\Delta q = A(q_d(t), \omega_d(t))\Delta_q\ +\ B(q_d(t), \omega_d(t))\Delta\omega_d \tag{2.23}$$

where,

$$\begin{aligned} A(q_d(t), \omega_d(t)) &= \begin{bmatrix} 0 & 0 & -\frac{r}{2}\sin\psi_d(t)(\omega_{dl}(t),\ \omega_{dr}(t)) \\ 0 & 0 & \frac{r}{2}\cos\psi_d(t)(\omega_{dl}(t),\ \omega_{dr}(t)) \\ 0 & 0 & 0 \end{bmatrix} \\ B(q_d(t), \omega_d(t)) &= \begin{bmatrix} \frac{r}{2}\cos\psi_d(t) & \frac{r}{2}\cos\psi_d(t) \\ \frac{r}{2}\sin\psi_d(t) & \frac{r}{2}\sin\psi_d(t) \\ \frac{r}{b} & -\frac{r}{b} \end{bmatrix} \end{aligned} \tag{2.24}$$

As the linearized system is a time-varying with the dependence on the time of the reference trajectory. Because of this the controllability analysis would be too involved. So as a special case a constant velocity is considered with $\psi_d(t) = \psi_{do}$, $\omega_{dl} = \omega_l$, $\omega_{dr} = \omega_r$. So the final A, B matrix will be a linear-time invariant system. As the rank of the controllability matrix is 3 from Eq. 2.25, the linearized system is controllable.

$$
\begin{aligned}
C &= \begin{bmatrix} B & AB & A^2B \end{bmatrix} \\
&= \begin{bmatrix}
\frac{r}{2}\cos\psi & \frac{r}{2}\cos\psi & -\frac{r^2}{2b}\sin\psi & \frac{r^2}{2b}\sin\psi & 0 & 0 \\
\frac{r}{2}\sin\psi & \frac{r}{2}\sin\psi & \frac{r^2}{2b}\cos\psi & -\frac{r^2}{2b}\cos\psi & 0 & 0 \\
\frac{r}{b} & -\frac{r}{b} & 0 & 0 & 0 & 0
\end{bmatrix} \\
&= rank(3)
\end{aligned}
\tag{2.25}
$$

Therefore from Sec. 2.2.1 and Sec. 2.2.2 we can say that the system is controllable about a point (or configuration) and about a trajectory.

CHAPTER 3

Trajectory Generation

3.1 Overview

In this work, trajectory design refers to generation of the position, velocity acceleration profiles corresponding to the given starting and the target position of the rover [1]. The problem of trajectory generation can be solved by fitting a necessary order polynomial between the desired initial and final configuration. Even though the problem of trajectory generation is well defined, the main problem occurs in trajectory continuity and the design of feasible trajectories such that the constraints of the system are not violated. Constraints of the system can be placed on the velocity, acceleration, path curvature etc. Various researchers have tackled the curvature constrained trajectory generation problem by considering a special polynomial that inherently have the properties of curvature continuity. An example of this kind of polynomial is the B-spline, which is a set of basis functions, that has these properties. B-spline polynomials have been used for online trajectory generation with curvature continuity [5]. The main drawback with B-spline polynomial is that it lacks a closed form analytical expression which makes it hard for finding a solution for constraining the velocity and acceleration along the trajectory.

Another example is the Bezier curve, which is also a basis function, and can be used for online trajectory generation. In-order to use bezier curves for the velocity and acceleration constrained trajectory generation, the coefficients of the polynomials have to be determined first. One way to solve for the coefficients(so that the final polynomial is constrianed upto required differentiable level) is to define an objective

function and solve for the coefficients [18]. One needs to be careful when working with bezier curves [1] as the polynomial is unstable for a polynomial order greater than 4 . Another drawback with bezier curves is that they lack curvature continuity between trajectory segments.

Some researchers have approached the problem of constrained trajectory generation by replacing the straight lines and circular arcs in dubins path [2] with a polynomial to constrain the velocity, acceleration. This method uses clothoid curves [19] instead of circular arcs so as to have continuous curvature, constrained velocity and acceleration trajectory.

One of the problems this thesis addresses is the problem of trajectory generation between a desired set of configurations, such that the acceleration along the trajectory does not exceed the maximum acceleration of the system.

3.2 Trajectory Generation with Acceleration Constraints

For the problem of constrained trajectory generation, a polynomial of sufficient order is selected to enforce the boundary conditions. The boundary conditions on position, velocity and acceleration is used to find the coefficients of the polynomials in terms of the total trajectory time t_f.

Let the position, velocity and acceleration of a point be,

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.1}$$

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \tag{3.2}$$

$$\ddot{q}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \tag{3.3}$$

[1] Bezier Curves, https://en.wikipedia.org/wiki/Bezier_curve

And let the initial positional coordinate be $q_{initial}$ and final target coordinate be q_{final}. Let's also assume that the rover starts with zero velocity, acceleration and ends up at the target coordinate q_{final} with zero final velocity, acceleration.

$$q(0) = q_{initial} \tag{3.4}$$

$$q(t_f) = q_{final} \tag{3.5}$$

$$\dot{q}(0) = \dot{q}(t_f) = 0 \tag{3.6}$$

$$\ddot{q}(0) = \ddot{q}(t_f) = 0 \tag{3.7}$$

where t_f = final time. The boundary conditions on the position, velocity and the acceleration of the rover are used to calculate the coefficients of the polynomials. The following results by substituting the above boundary conditions into Eqs. 3.1, 3.2 and 3.3,

$$a_0 = q(0) \tag{3.8}$$

$$a_1 = 0 \tag{3.9}$$

$$a_2 = 0 \tag{3.10}$$

$$q_{final} - q_{initial} = a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \tag{3.11}$$

$$0 = 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \tag{3.12}$$

$$0 = 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 \tag{3.13}$$

Solving Eqs. 3.11, 3.12 and 3.13, the coefficients a_3, a_4, a_5 are obtained in terms of the boundary conditions and final time, t_f as,

$$a_3 = 10\left(\frac{\Delta q}{t_f^3}\right) \tag{3.14}$$

$$a_4 = -15\left(\frac{\Delta q}{t_f^4}\right) \tag{3.15}$$

$$a_5 = 6\left(\frac{\Delta q}{t_f^5}\right) \tag{3.16}$$

where $\Delta q = q_{final} - q_{initial}$. The acceleration of the polynomial at each time is constrained by the maximum attainable acceleration of the system by,

$$|\ddot{q}(t)| \leq a_{max} \tag{3.17}$$

where a_{max} is the maximum acceleration. Eq. 3.17 is solved to obtain the final time of trajectory. So the problem now becomes,

$$\begin{aligned} &minimize: \ \frac{1}{2}\ddot{q}^2(t) \\ &subject\ to: \ |\ddot{q}(t)| \leq |a_{max}|\,. \end{aligned} \tag{3.18}$$

By defining $\tau = \frac{t}{t_f}$, $\ddot{q}(t)$ can be written as,

$$\ddot{q}(t) = \left| 60 \left(\frac{\Delta q}{t_f^2} \right) \left[\tau(1 - 3\tau + 2\tau^2) \right] \right| \tag{3.19}$$

Using the necessary conditions for finding the minimum of a function, we get

$$\frac{d(\ddot{q}(t))}{d\tau} = 0 \tag{3.20}$$

Then solution for the variable τ can be obtained as

$$\tau = \frac{1}{2} \pm \frac{1}{\sqrt{12}} \tag{3.21}$$

By substituting Eq. 3.21 in Eq. 3.19 the final time t_f is obtained in terms of the maximum acceleration and the boundary conditions as,

$$t_f \geq \sqrt{\frac{10}{\sqrt{3}} \frac{\Delta q}{a_{max}}} \tag{3.22}$$

Therefore, the values for a_3, a_4 and a_5 can be obtained by substituting t_f into respective equations.

The resulting optimal trajectory is called Minimum-Jerk trajectory as we are equating the jerk (derivative of the acceleration) to zero in Eq. 3.20.

3.3 Simulation Based Verification of Derived Trajectory

In-order to verify the derived analytical result from Eq. 3.22 for the final trajectory time, a maximum acceleration of 0.5 m/sec^2 is selected to generate the reference trajectory from an initial position of $(10, 2)$ to the final position $(17, 15)$. The final time obtained from Eq. 3.22 is about 17.25 *sec*. The reference trajectory that was generated after calculating coefficients by using Eqs. 3.14, 3.15 and 3.16 is shown in Fig. 3.1.

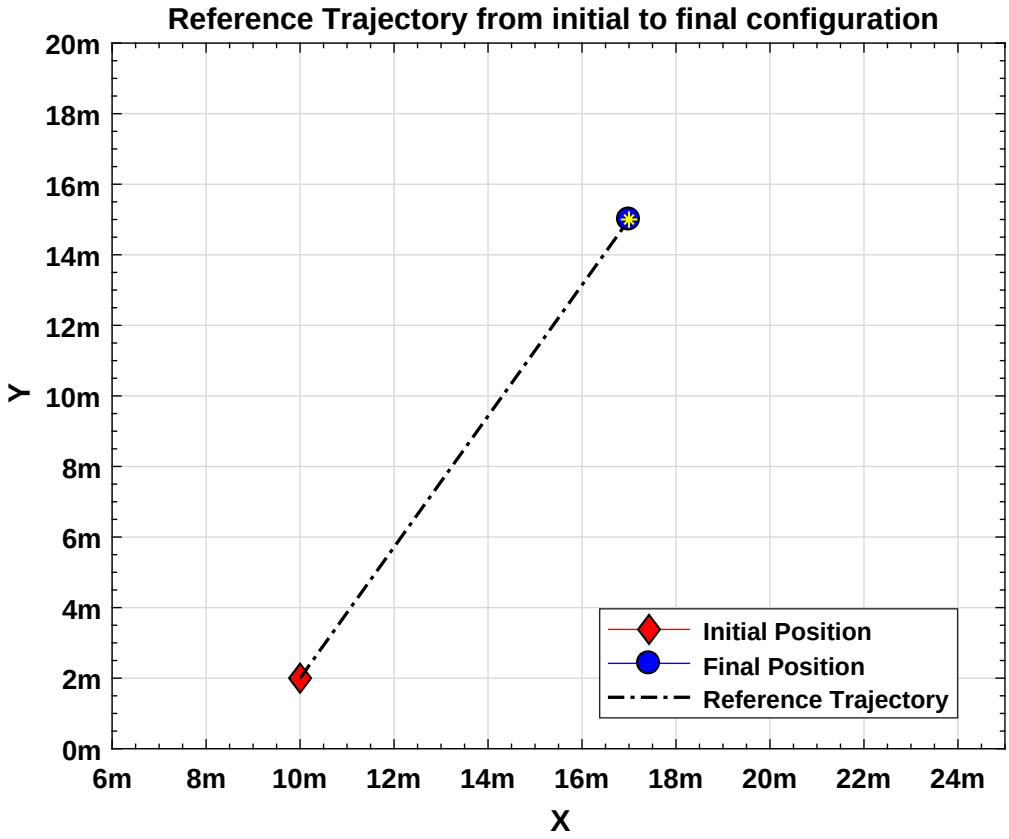


Figure 3.1: Reference Trajectory from an initial position$(10, 2)$ to final position $(17, 15)$

The acceleration profiles of the 5^{th} order polynomial in individual x and y-directions is shown in Figs. 3.2, 3.3. From Figs. 3.2 and 3.3 it can be verified that the acceleration of the polynomial is below the specified maximum acceleration which is 0.5 m/sec^2. And also from Fig. 3.4 it can be verified the resultant reference acceleration is below the specified acceleration bound which is 0.5 m/sec^2.

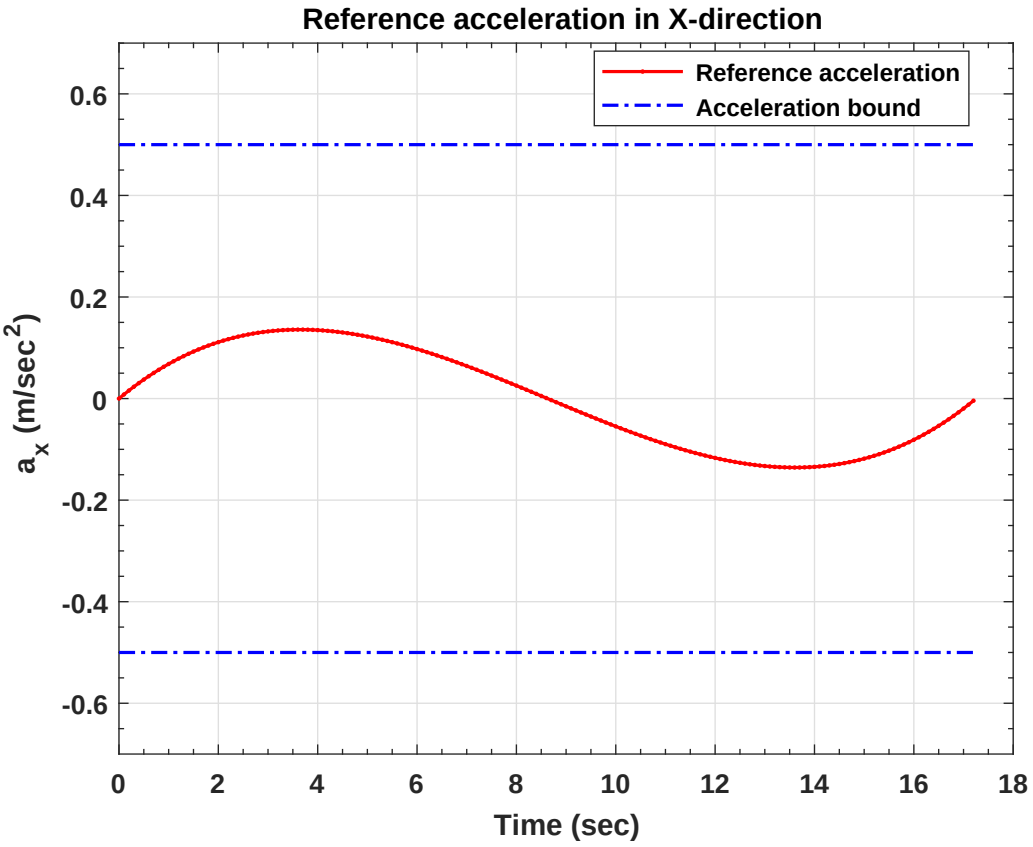


Figure 3.2: Reference acceleration in X-direction (m/sec^2) with a maximum acceleration bound of 0.5 m/sec^2

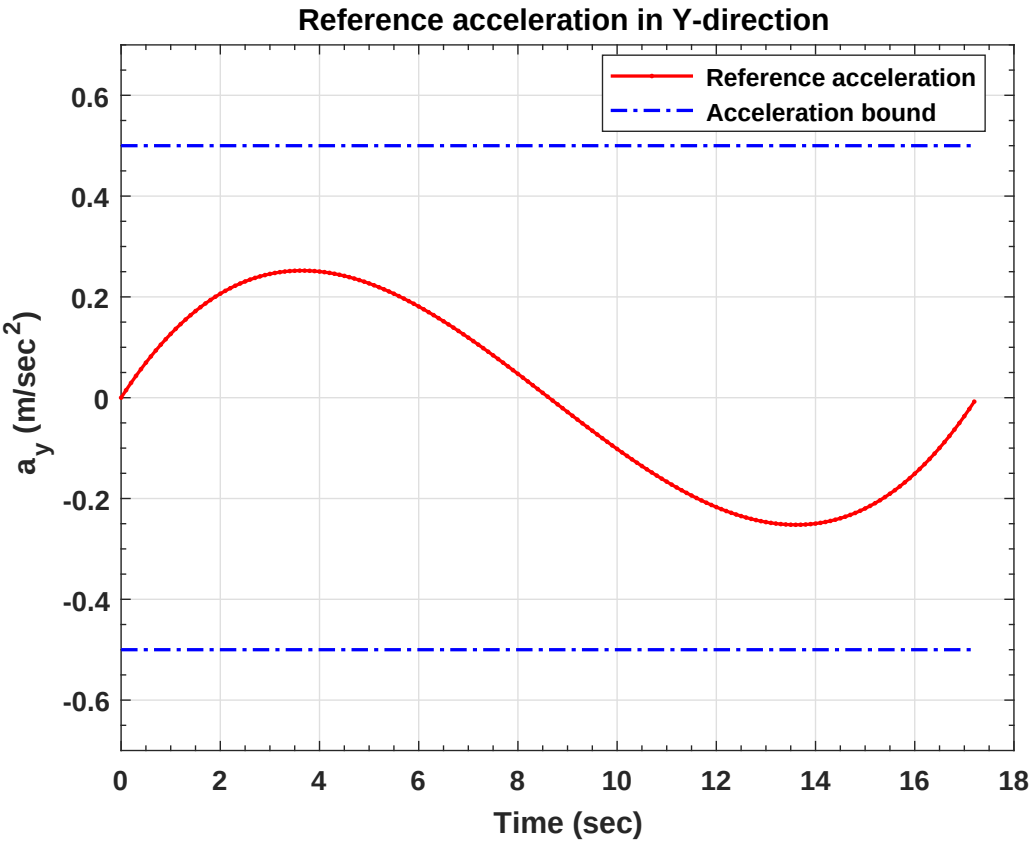


Figure 3.3: Reference acceleration in Y-direction (m/sec^2) with a maximum acceleration bound of 0.5 m/sec^2

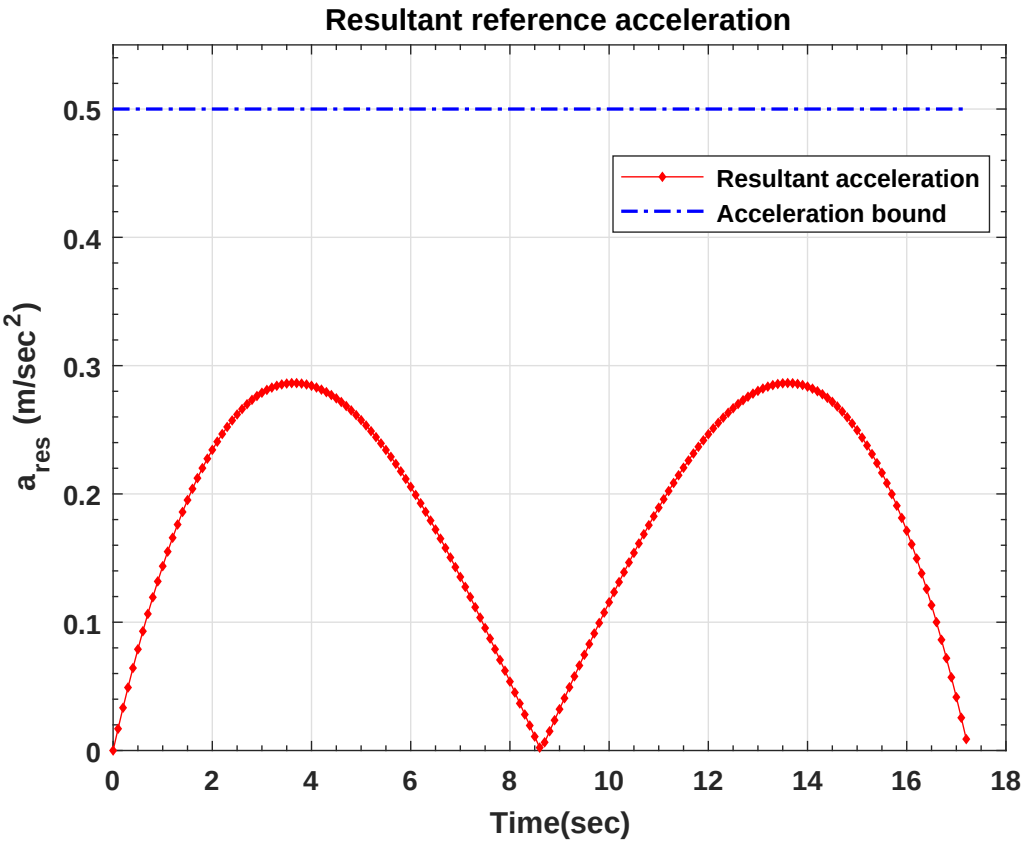


Figure 3.4: Resultant reference acceleration (m/sec^2) with a maximum acceleration bound of 0.5 m/sec^2

where a_{res} is the resultant acceleration and is given as $\sqrt{a_x^2 + a_y^2}$.

Therefore from the above figures it is verified that the result given in Eq. 3.22 can be used for generating trajectory with acceleration constraint.

3.4 Circular Trajectory Stitching for Turning

As the constrained acceleration trajectory that was created in previous section does not take into account the final heading angle of the rover, a circular trajectory with a constant resultant velocity along the circular path is created to align the robot towards the next target waypoint. In-order to create the circular trajectory,

the current heading (ψ_0) and the final desired heading (ψ_f) is considered. The final heading angle is calculated by taking the tangent angle between the current waypoint and the next waypoint. Fig. 3.5 shows the different angles that are extracted from the current and final heading angle.

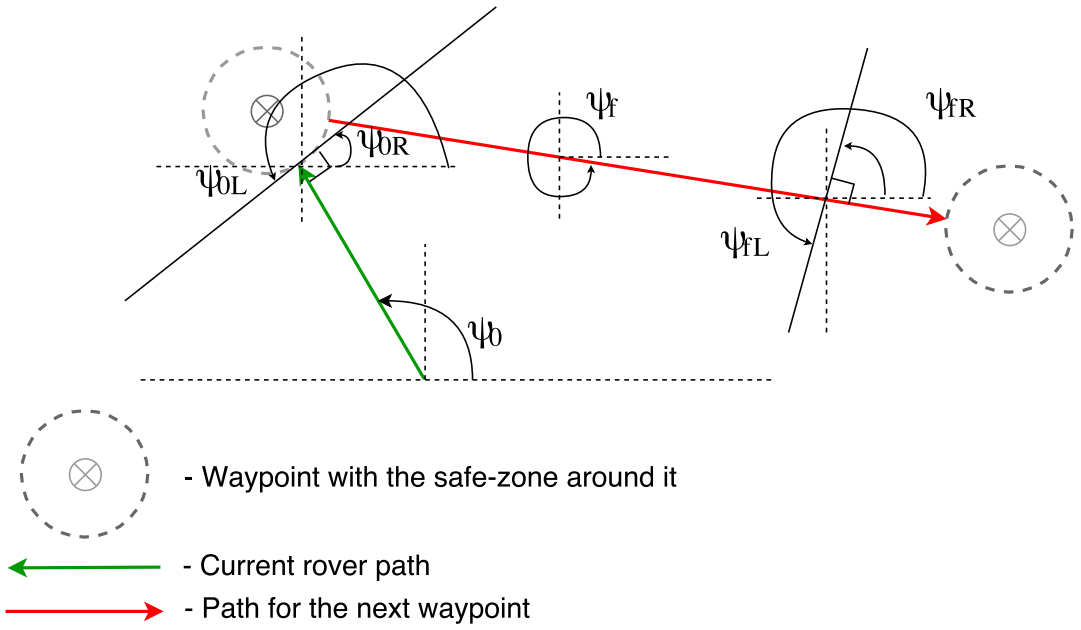
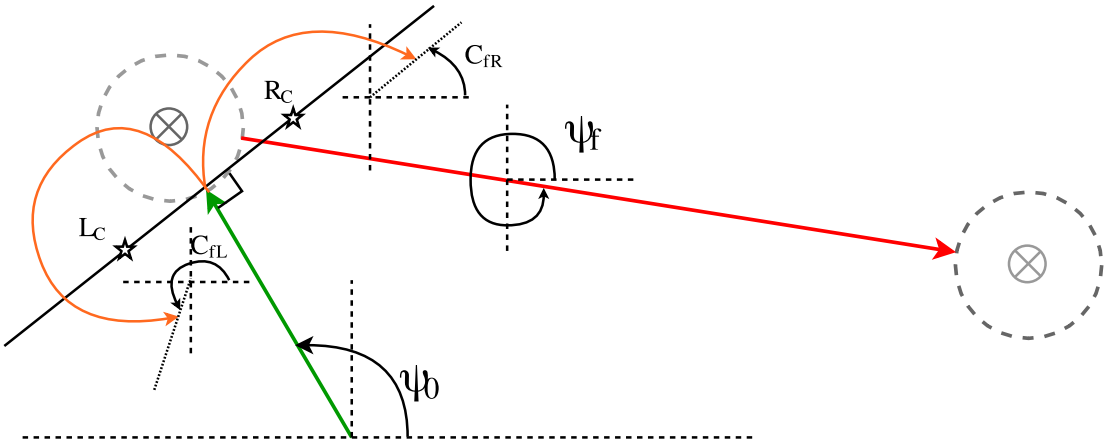


Figure 3.5: Angles that were extracted by using current and final heading angle for circular trajectory stitching

where ψ_{0R}, ψ_{0L} are the angles of the line that is perpendicular to the current rover path in right and left directions respectively. ψ_{fR}, ψ_{fL} are the angles of the line that is perpendicular to the path for the next waypoint in right and left directions respectively.

Once these angles are calculated, the starting angle, ending angle and the center for the circle is calculated based on the direction of the turn as shown in Fig. 3.6.



L_C - Center of the circle for the left turning

R_C - Center of the circle for the right turning

Figure 3.6: Circular path creation based on the extracted angles

where C_{fL} is the ending angle of the circle for the case of turning towards left direction and C_{fR} is the ending angle of the circle for the case of turning towards right direction with starting angle of the circle, C_0 for both the cases being the current heading angle (ψ_0) of the rover.

Different angles and parameters of the circle that are shown in Figs. 3.5, 3.6 are calculated as follows,

$$\begin{aligned}
\psi_{0L} = \psi_{0R} = \psi_{0T} &= \alpha + \psi_0 - \frac{\pi}{2} \\
\psi_{fR} = \psi_{fL} = \psi_{fT} &= \psi_f - \gamma\frac{\pi}{2} \\
C_{fR} = C_{fL} = C_f &= \gamma(\psi_{fT} - \psi_{0T}) \\
L_{Cx} = R_{Cx} = X_c &= X_{current} + R_{min}\cos(\pi - \alpha + \psi_0 - \frac{\pi}{2}) \\
L_{Cy} = R_{Cy} = Y_c &= Y_{current} + R_{min}\sin(\pi - \alpha + \psi_0 - \frac{\pi}{2}) \\
C_{0R} &= \psi_{fT} \\
C_{0L} &= \psi_{0T}
\end{aligned} \tag{3.23}$$

where γ is defined as -1 for the right turn (or) +1 for the left turn, α is defined as 0 for the left turn (or) π for the right turn, C_{0R} is the starting angle of the circle for the right turn, C_{0L} is the starting angle of the circle for the left turn, R_{min} is the minimum turning radius, ($X_{current}$, $Y_{current}$) is the current position of the rover.

By using the angles and the parameters of the circle defined in Eq. 3.23 and from Eq. 3.24, a circular trajectory is generated and stitched between consecutive minimum-jerk trajectory segments as shown in Fig. 3.7

$$
\begin{aligned}
X(t) &= X_c + R_{min}\cos\left(C_f\frac{t}{t_f} + C_0\right) \\
Y(t) &= Y_c + R_{min}\sin\left(C_f\frac{t}{t_f} + C_0\right) \\
\dot{X}(t) &= -\gamma R_{min}C_f\frac{t}{t_f}\sin\left(C_f\frac{t}{t_f} + C_0\right) \\
\dot{Y}(t) &= \gamma R_{min}C_f\frac{t}{t_f}\cos\left(C_f\frac{t}{t_f} + C_0\right) \\
\ddot{X}(t) &= -R_{min}\left(C_f\frac{t}{t_f}\right)^2\cos\left(C_f\frac{t}{t_f} + C_0\right) \\
\ddot{Y}(t) &= -R_{min}\left(C_f\frac{t}{t_f}\right)^2\sin\left(C_f\frac{t}{t_f} + C_0\right)
\end{aligned}
\tag{3.24}
$$

where C_0, is selected as C_{0L} for left turn (or) C_{0R} for right turn, t_f, is calculated based on the resultant constant velocity(V_{const}) along the path and is given as $t_f = \frac{R_{min}C_f}{V_{const}}$.

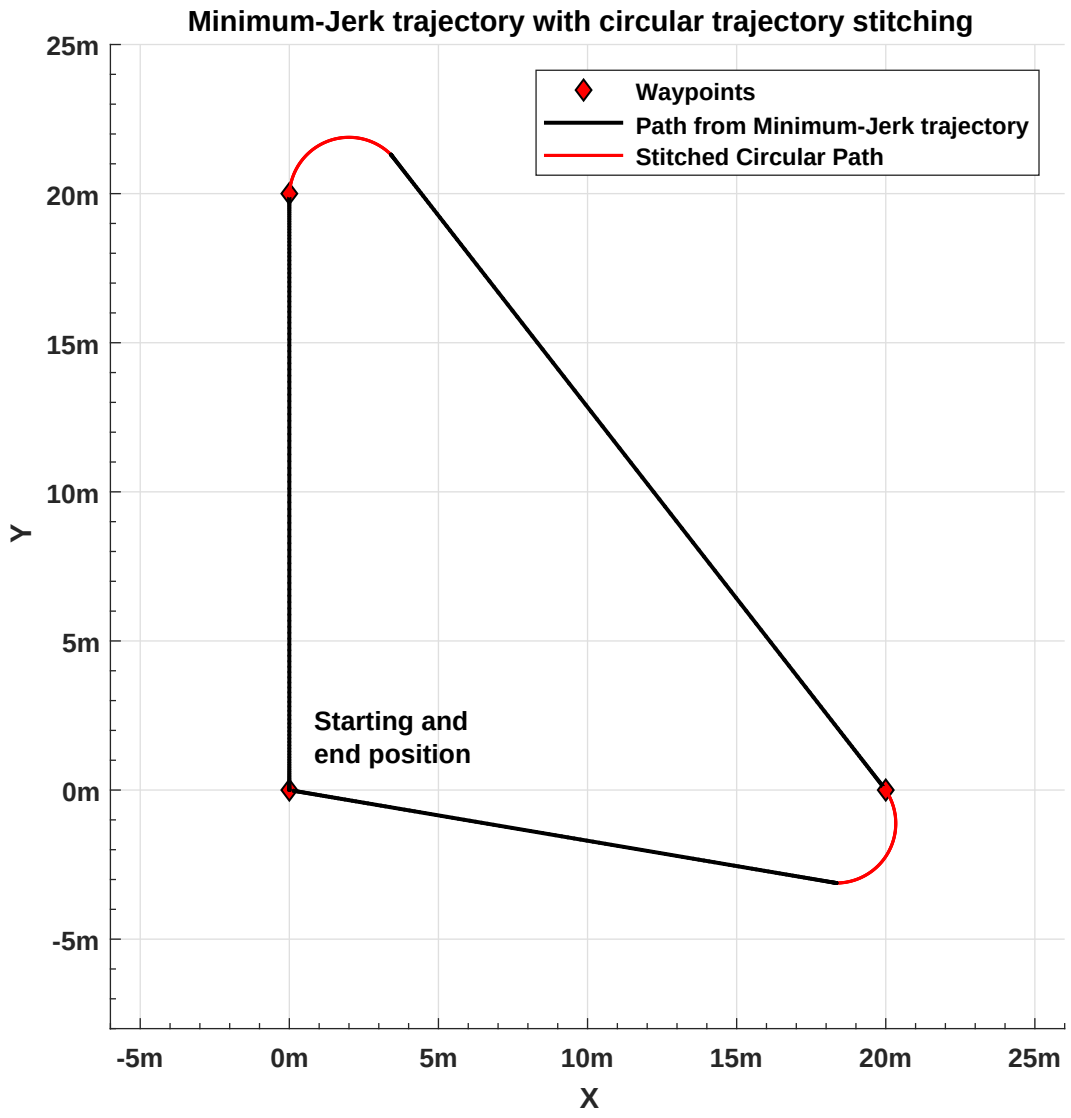


Figure 3.7: Reference path generated by stitching circular segments to minimum-jerk trajectory with a turn radius of 2 m and a constant velocity of 0.5 m/sec for the circular segments

CHAPTER 4

Control Law

4.1 Overview

Traditionally, the control of a mobile robot has been designed by either using point stabilization [20] or by redefining the problem as a path/trajectory tracking problem [21]. The path/trajectory tracking control problem is more appropriate, as the control objectives(time of travel, fuel consumption) can be incorporated into the path-planning procedure [22, 23]. Many control algorithms are proposed in path/trajectory tracking framework such as PID [24], Lyapunov-based controller design [25], feedback-linearization technique [16], model predictive controller [26], backstepping controller[10]. Some of the approaches only guarantee local stability, while others ensure global stability and convergence under certain assumptions.

Kinematic model based control design is more appropriate if the associated actuator dynamics does not contain a noticeable amount of lag. At the same time dynamic model based control design is efficient when there is a sufficient lag in the actuators. In this project a kinematic model is considered for controller design/guidance law.

4.2 The idea of Backstepping

Backstepping is a recursive design technique that gives us the ability to construct both the feedback control laws and associated Lyapunov functions in a nice systematic manner. The main difference between feedback linearization technique and the backstepping is that the feedback linearization method cancels all nonlinearities in the system. The backstepping method gives the designer the flexibility to

exploit the so-called "good" nonlinearities while cancelling the "bad" nonlinearities that destabilizes the system. This is very important as the cancellation of all nonlinearities requires precise system models which are very difficult to obtain.

The key idea of backstepping [27], [28] can be explained by considering a "strict-feedback form" type system:

$$\dot{x}_1 = f_1(x_1, x_2) \tag{4.1}$$

$$\dot{x}_2 = x_3 + f_2(x_1, x_2) \tag{4.2}$$

$$\dot{x}_3 = u + f_3(x_1, x_2, x_3) \tag{4.3}$$

The first step is to stabilize x_1 at 0. Let's consider x_2 as a virtual control to stabilize x_1 with the feedback law $x_2 = \alpha_1(x_1)$. Let's also denote that $z_1 = x_1$. So the Eq. 4.1 now becomes,

$$\dot{z}_1 = f_1(z_1, \alpha_1(x_1)) \tag{4.4}$$

Now, the designer needs to choose the stabilizing function $\alpha_1(x_1)$ such that x_1 is stable. The function $\alpha_1(x_1)$ can be chosen by considering a Lyapunov function candidate for the system $\dot{z_1}$,

$$V_1(x_1) = \frac{1}{2} z_1^2 \tag{4.5}$$

So, in-order for the system to be considered as stable the derivative of the Lyapunov function for the corresponding system needs to be negative-definite.

$$\dot{V}_1 = z_1 \dot{z}_1 \tag{4.6}$$

$$= z_1 f_1(z_1, \alpha_1(x_1)) \tag{4.7}$$

The stabilizing function $\alpha_1(x_1)$ needs to be chosen such that $\frac{\partial V_1}{\partial z_1} f_1(z_1, \alpha_1(x_1)) < 0$. However we can achieve $x_2 = \alpha_1(x_1)$ only with an error of $z_2 = x_2 - \alpha_1(x_1)$. Therefore, Eq. 4.1, 4.2 now becomes,

$$\dot{z}_1 = f_1(z_1, \alpha_1(z_1)) + z_2 \Psi_1(z_1, z_2) \tag{4.8}$$

$$\dot{z}_2 = x_3 + f_2(z_1, z_2 + \alpha_1(z_1)) - \dot{\alpha}_1 \tag{4.9}$$

where $\Psi_1(z_1, z_2)$ in Eq. 4.8 is obtained by expressing f_1 as,

$$f_1(z_1, z_2 + \alpha_1(z_1)) = f_1(z_1, \alpha_1(z_1)) + z_2 \Psi_1(z_1, z_2) \tag{4.10}$$

It should also be noted that $\dot{\alpha}_1$ is also known explicitly,

$$\dot{\alpha}_1 = \frac{\partial \alpha}{\partial x_1} \dot{x}_1 \tag{4.11}$$

$$= \frac{\partial \alpha}{\partial x_1} f_1(x_1, x_2) \tag{4.12}$$

$$= \frac{\partial \alpha}{\partial z_1} f_1(z_1, z_2 + \alpha_1(z_1)) \tag{4.13}$$

In the second step we will now use state x_3 as a virtual control to drive the $f_2(x_1, x_2)$ system to 0 with a feedback law $x_3 = \alpha_2(z_1, z_2)$. Now, again the the stabilizing function α_2 need to be determined so as to satisfy the Lyapunov stability for the system z_2. The Lyapunov function candidate be,

$$V_2 = V_1(z_1) + \frac{1}{2} z_2^2 \tag{4.14}$$

Now, we want to make $\dot{V}_2$ negative-definite,

$$\begin{aligned} \dot{V}_2 &= \frac{\partial V_1}{\partial z_1} f_1(z_1, \alpha_1(z_1)) \\ &+ z_2 \left(\frac{\partial V_1}{\partial z_1} \Psi_1(z_1, z_2) + \alpha_2 + f_2(z_1, z_2 + \alpha_1(z_1, z_2)) \right) \end{aligned}$$

As the first term in the above equation was made negative in step 1, we only need to choose α_2 to make the expression negative. So, by choosing the α_2 as,

$$\alpha_2 = -z_2 - \frac{\partial V_1}{\partial z_1} \Psi_1(z_1, z_2) - f_2(z_1, z_2 + \alpha_1(z_1)). \tag{4.15}$$

the expression for $\dot{V}_2$ now becomes,

$$\dot{V}_2 = \frac{\partial V_1}{\partial z_1} f_1(z_1, \alpha_1(z_1)) - z_2^2 \tag{4.16}$$

Again, since we cannot achieve $x_3 = \alpha_2(z_1, z_2)$ perfectly, there is an error $z_3 = x_3 - \alpha_2(z_1, z_2)$ and the expression for $\dot{V}_2$ now becomes,

$$\dot{V}_2 = \frac{\partial V_1}{\partial z_1} f_1(z_1, \alpha_1(z_1)) - z_2^2 + z_2 z_3 \tag{4.17}$$

Finally we know the relationship between $z_1,\ z_2,\ z_3$ and $x_1,\ x_2,\ x_3$. Therefore, the system can be written as

$$\dot{z}_1 = f_1(z_1, \alpha_1(z_1)) + z_2 \Psi(z_1, z_2) \tag{4.18}$$

$$\dot{z}_2 = z_3 + \alpha_2(z_1, z_2) + f_2(z_1, z_2 + \alpha_1(z_1)) - \dot{\alpha}_1 \tag{4.19}$$

$$\dot{z}_3 = u + f_3(z_1, z_2 + \alpha_1(z_1), z_3 + \alpha_2(z_1, z_2)) - \dot{\alpha}_2 \tag{4.20}$$

In the final step, no virtual control is needed as the actual control(u) is available. Now, a feedback law for u need to be selected to make the derivative of the Lyapunov function for the third cascaded system($\dot{z}_3$) be negative-definite

$$V_3 = V_2 + \frac{1}{2} z_3^3 \tag{4.21}$$

$$\dot{V}_3 = \frac{\partial V_1}{\partial z_1} f_1(z_1, \alpha_1(z_1)) - z_2^2 + z_2 z_3 + z_3(u + f_3(z_1, z_2, z_3, \alpha_1, \alpha_2) - \dot{\alpha}_2) \tag{4.22}$$

So the control law u is selected as

$$u = -z_3 - f_3 + \dot{\alpha}_2 - z_2 \tag{4.23}$$

this guarantees that $\dot{V}_3 < 0$ for all nonzero $z_1,\ z_2,\ z_3$. This results in global asymptotic stability property with equilibrium at 0. This designed feedback law can now be expressed as a function of $x_1,\ x_2,\ x_3$.

From the above derivation it can be seen that the design of a feedback control

law by using the backstepping control law tells us about the stability of the system in the design phase itself. So, the designer can use this idea to exploit the desired behavior that's needed. The above example of strict-feedback form is only used to motivate the discussion and the explained backstepping control law can be used for any type of system that's not necessarily of strict-feedback form and not necessarily be feedback linearizable.

4.3 Design of nonlinear control-law for differential-drive robot

The kinematic model of the rover is illustrated in Fig. 2.1. It is evident that the only way to control the position and orientation of the robot is by changing the wheel speed. So, the low level controllers on the rover will directly command wheel RPM based on a PWM voltage input.

Let the error in position in both x and y directions be,

$$e_x = x_{act} - x_{ref} \tag{4.24}$$

$$e_y = y_{act} - y_{ref} \tag{4.25}$$

where subscript (act) represents actual state of the rover and subscript (ref) represents reference trajectory.

Let the time derivative of the position tracking errors are taken as

$$\dot{e}_x = \dot{x}_{act} - \dot{x}_{ref} \tag{4.26}$$

$$\dot{e}_y = \dot{y}_{act} - \dot{y}_{ref} \tag{4.27}$$

The desired heading angle and velocity signals, ψ_{des} and $\hat{v}$ respectively, are defined such that there is exponentially stable position tracking errors, i.e.

$$\dot{e}_x = -\lambda_x e_x \tag{4.28}$$

$$\dot{e}_y = -\lambda_y e_y \tag{4.29}$$

for any $\lambda_x > 0$, and $\lambda_y > 0$. It follows that

$$\begin{aligned} \hat{v}\cos\left(\psi_{des}\right) &= \dot{x}_{ref} - \lambda_x e_x \\ \hat{v}\sin\left(\psi_{des}\right) &= \dot{y}_{ref} - \lambda_y e_y. \end{aligned} \tag{4.30}$$

Then, by using the information in Eq. 4.30, the desired heading and velocity signals for exponentially stable position tracking are defined as

$$\psi_{des} = \tan^{-1}\left(\frac{y_{ref} - \lambda_y e_y}{x_{ref} - \lambda_x e_x}\right), \tag{4.31}$$

$$\hat{v} = \sqrt{\left(\dot{x}_{ref} - \lambda_x e_x\right)^2 + \left(\dot{y}_{ref} - \lambda_y e_y\right)^2}. \tag{4.32}$$

It should be noted that Eq. 4.32 for the desired translational velocity, $\hat{v}$, can be written in terms of the wheel speed commands as

$$\hat{v} = \frac{r}{2}\left(\omega_L + \omega_R\right).$$

Therefore, an additive relation for the wheel commands is given by

$$\omega_L + \omega_R = \frac{2}{r}\sqrt{\left(\dot{x}_{ref} - \lambda_x e_x\right)^2 + \left(\dot{y}_{ref} - \lambda_y e_y\right)^2}. \tag{4.33}$$

It is also desired that the heading angle tracking error, $e_\psi = \psi_{act} - \psi_{ref}$, be asymptotically stable. In other words, it is prescribed that $\dot{e}_\psi = -\lambda_\psi e_\psi$. The time derivative for the heading angle tracking error is given by $\dot{e}_\psi = \dot{\psi} - \dot{\psi}_d$. Combining the kinematic equations in Eq. 2.14 with the heading angle tracking error derivative and the prescribed condition leads to another equation in terms of the wheel commands as

$$\omega_R - \omega_L = \frac{b}{r}\left(\dot{\psi}_d - \lambda_\psi e_\psi\right). \tag{4.34}$$

The $\dot{\psi}_{des}$ term can be derived by taking the time derivative of Eq. 4.31, which leads to

$$\dot{\psi}_{des} = \frac{1}{\hat{v}}\left[\cos\left(\psi_{des}\right)\left(\ddot{y}_{ref} - \lambda_y\left(\hat{v}\sin\left(\psi\right) - \dot{y}_{ref}\right)\right) - \sin\left(\psi_{des}\right)\left(\ddot{x}_{ref} - \lambda_x\left(\hat{v}\cos\left(\psi\right) - \dot{x}_{ref}\right)\right)\right] \tag{4.35}$$

where the $\cos(\psi_{des})$ and $\sin(\psi_{des})$ terms come from the formulation in Eq. 4.30.

Solving Eq. 4.33 and 4.34 simultaneously leads to the following functions for the left and right wheel speeds as

$$\begin{aligned} \omega_R &= \frac{1}{2}\left[\frac{2}{r}\hat{v} + \frac{b}{r}\left(\dot{\psi}_{des} - \lambda_\psi e_\psi\right)\right], \\ \omega_L &= \frac{1}{2}\left[\frac{2}{r}\hat{v} - \frac{b}{r}\left(\dot{\psi}_{des} - \lambda_\psi e_\psi\right)\right], \end{aligned}$$

It is important to note that as $\hat{v}$ approaches zero in Eq. 4.35, the value for $\dot{\psi}_d$ is not defined. In order to avoid this, an algorithm for singularity avoidance is employed. In general, the value of $\dot{\psi}_d$ is defined to be zero if $\hat{v} \leq \epsilon$, for some $\epsilon \ll 1$. Thus, the algorithm assumes that ψ_d is held constant as a reference value based on which waypoints the rover is traveling in-between. Formally, the following algorithm is used to modify the wheel input speeds and avoid the singularities.

if (t $>$ 0) and ($\hat{v} \leq \epsilon$) **then**

$$\omega_L + \omega_R = \frac{2}{r}\sqrt{(\dot{x}_{ref} - \lambda_x e_x)^2 + (\dot{y}_{ref} - \lambda_y e_y)^2}$$

$$\psi_{ref} = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right)$$

$$\omega_R - \omega_L = -\lambda_k(\psi - \psi_{ref})$$

end if

A Lyapunov-stability analysis for the proposed control law is given in Appendix A

4.3.1 Control-Gain Selection

As the proposed controller is based on the exponential decay of error which in-turn dependent on the control gains ($\lambda_x,\ \lambda_y,\ \lambda_\psi$) by choosing appropriate control gains good trajectory tracking can be achieved. A simulation based gain selection and its performance is already shown in [29]. But when an experimental setup is considered for tracking the reference trajectory, a static gain is not sufficient for

bounded error tracking. Because of the cyber-physical system architecture, there will be network communication delay which results in data loss and communication blackouts. So there will be an unbounded increase in error. In-order to reduce this error, a relationship between the gains and error is desired, where an increase in error will increase the gain associated with it to preserve the exponential error decay. With that in mind, Eq. 4.36 gives a relationship between the error and associated gain,

$$\dot{e} = -\lambda e. \tag{4.36}$$

The differential equation in Eq. 4.36 is solved and the following relation is obtained

$$\lambda(t) = \frac{\ln\left(\frac{e(t)}{e_0}\right)}{t - t_0}, \tag{4.37}$$

where e_0 is the initial condition error and t_0 is the initial time.
As the rover starting position is considered as the origin(0,0) for the local reference frame and the reference trajectory is created from the origin, Eq. 4.37 is modified to,

$$\lambda(t) = k \ln\left(\frac{|e(t)|}{0.1} + 0.1\right). \tag{4.38}$$

Therefore, Eq. 4.38 is used to calculate the required gain based on the error. The value for k is selected by doing some experiments on the rover vehicle. The constant k selected were $k_x = 0.1,\ k_y = 0.1,\ k_\psi = 2$.

CHAPTER 5

Experimental Setup

To test the minimum-jerk trajectory from Chap. 3 and the modified back-stepping controller from Chap. 4 an experimental differential-drive robot was built from the ground-up. The final assembly of the rover is shown in Fig. 5.1. The differential-drive robot is equipped with several sensors for navigation and to sense it's environment. Together, these sensors give the rover the ability to test the derived algorithms in real-time in a constrained environment.

Figure 5.1: Differential-drive robot built at Aerospace Systems Laboratory, UTA

5.1 On-board Sensors

The differential-drive robot shown in Fig. 5.1 is equipped with different sensors to localize the rover position in either a global reference frame or a local reference frame. These sensors are explained in following subsections.

5.1.1 Pixhawk

Pixhawk is an high-performance autopilot suitable for any robotic platform that can move. The pixhawk internally contains autopilot software which can be used for autonomous navigation. At the same time, the data from the sensors inside pixhawk can be accessed and routed to other software modules. This makes it ideal for testing custom algorithms on the fly without re-writing the internal pixhawk firmware. Pixhawk internally contains a 3-axis gyroscope, 3-axis accelerometer, magnetometer and different types of communication ports [30].

Figure 5.2: Pixhawk-Autopilot used on the ASL-Gremlin-Rover
Source: https://pixhawk.org/modules/pixhawk

5.1.2 GPS/Compass

The GPS (Global Positioning System) sensor is a device which gives the current position of the user/robot on earth in a global frame, in terms of Latitude, Longitude, and Altitude on Earth.

The Compass is a device which gives the current bearing angle of the user with respect to the magnetic north of the Earth at the current position. Together GPS and Compass is an indispensable tool when it comes to navigation of an robot in outdoor environments.

The purpose of the GPS and compass in this project is to get the global position of the rover and also it's current bearing angle. This information helps us to localize the rover in a local reference frame.

Figure 5.3: 3DR u-blocx GPS/Compass used on the ASL-Gremlin-Rover
Source: http://ardupilot.org/copter/docs/common-installing-3dr-ublox-gps-compass-module.html

5.1.3 Micro-Controller

A micro-controller is a small computer on a single integrated circuit. Micro-controllers can be used for digital signal processing and voltage manipulation to control the real hardware. In the case of a differential-drive robot, the micro-controller is used for converting the commanded PWM(Pulse-Width-Modulation) signal to the

motor input voltage which in-terms controls the speed of the motor. The micro-controller used on our experimental rover is from the Arduino stack, which is shown below.

Figure 5.4: Arduino-Mega micro-controller used on the ASL-Gremlin-Rover
Source: https://store.arduino.cc/usa/arduino-mega-2560-rev3

5.1.4 Motors

A high-power, 6V brushed DC motor combined with a gearbox and a quadrature encoder on the motor shaft is used to rotate the wheels. The quadrature encoder attached on the motor shaft is used to calculate the angular velocity of the wheel. This helps us to localize the rover based on the technique of dead-reckoning.

The main disadvantage of the dead reckoning method is that the position information of the rover will drift because of wheel-slippage, wheel-tire flattening etc.

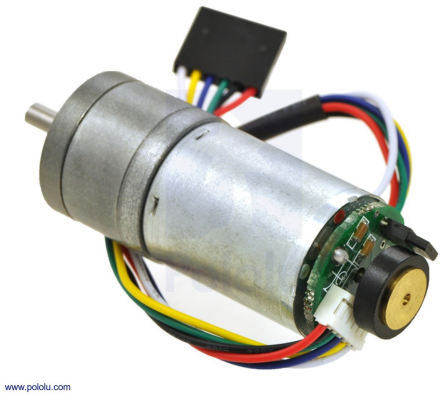


Figure 5.5: Brushed DC Motor with quadrature encoder used on the ASL-Gremlin-Rover

Source: https://www.pololu.com/product/2275

5.1.5 Odroid-XU4

The Odroid-XU4 is a high-performance, on-board computer which is used by researchers all over the world for online image processing, hardware interfacing, etc. On this rover, the Odroid-XU4 is used to collect the sensor data from pixhawk autopilot, send commanded voltage signals to the arduino micro-controller, etc.

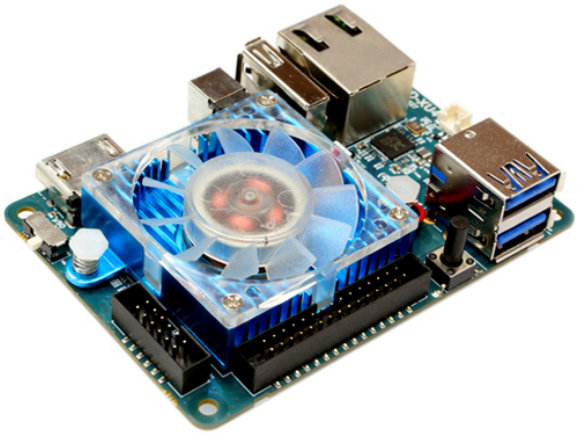


Figure 5.6: On-board computer on ASL-Gremlin-Rover

Source: http://www.hardkernel.com/main/products/`prdt_info`.php

5.2 Data Transfer Between Rover Components

Once all the necessary components are connected to pixhawk and the sensors are calibrated based on the instructions given on the ardupilot website[1], the next step is to get the sensor data from the pixhawk to the on-board computer. The sections below describes the data transfer configuration between different sensors and components on-board the rover. The software framework called Robot Operating System (ROS) [31], is used to setup the data transfer between the components and explained in Appendix B.

5.2.1 Communication Setup between Pixhawk and On-board Computer

There are two ways to get the sensor data from Pixhawk to the on-board computer. One way is to use the Serial port or the Telemetry port of the Pixhawk and write a serialization software module to re-route the data from the sensors to the computer. This requires the knowledge of writing serial software modules. Another way is to use the Mavlink enabled software packages that already performs the serialization with minimalistic configuration. In this work the second approach is used. The reason to use the Mavlink enabled software packages is that, Pixhawk internally uses the Mavlink supported firmware which reduces incompatible message type errors. The package called MAVROS [32], which is an open-source software package from ROS (Robot Operating System) is designed to communicate between Pixhawk and the computer that the Pixhawk is connected. In-order to send(or get) the data from the Pixhawk to Odroid, the **Telem2** port of the Pixhawk needs to be connected to the Odroid via a FTDI-cable. Even though the Pixhawk's USB port can be used for data transfer, the use of the USB port is not suggested because of data loss and connection loss while the Pixhawk is in operation. The connection between Odroid

[1] Ardupilot, http://ardupilot.org/copter/docs/common-pixhawk-wiring-and-quick-start.html

and Pixhawk is shown in Fig. 5.7. After the wiring is done, the setup, given in Appendix B.1 can be used to get the data from Pixhawk to Odroid.
Once the setup is done, all the Pixhawk sensor data is available in the form of ROS topics, which can be used to access the data.

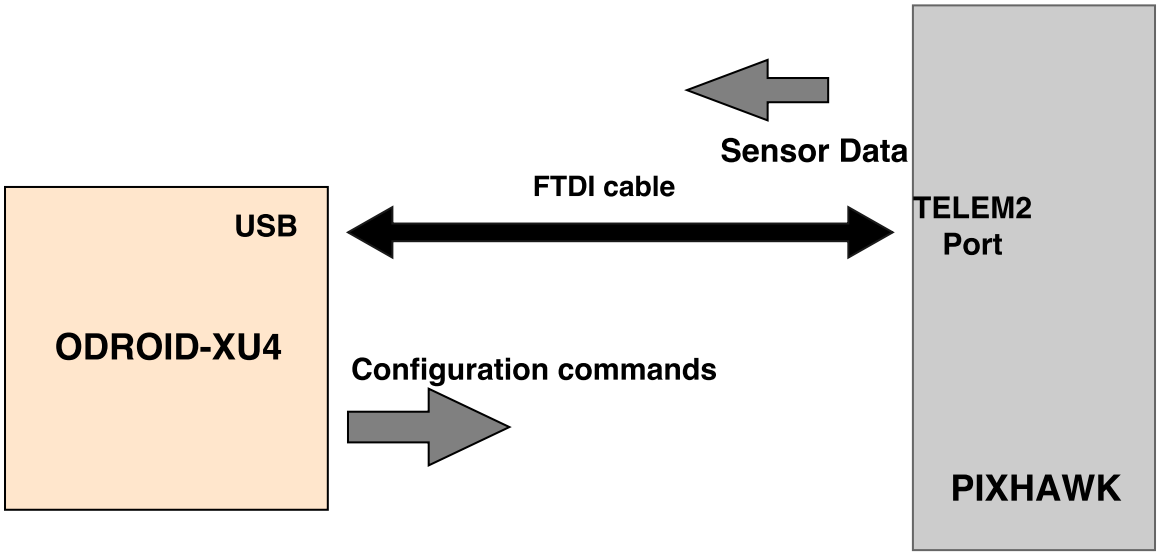


Figure 5.7: Connection between pixhawk and odroid that was used on ASL-Gremlin-Rover

5.2.2 Communication Setup between Arduino and On-board Computer

Data from the Arduino can be transferred to the Odroid by opening a serial communication from arduino. This way, data from different sensors connected to the Arduino can be accessed by the Odroid and at the same time data can be sent to the Arduino from the computer. Since the ROS framework is being used for data transfer, it'll be helpful if our Arduino is also registered with the ROS architecture. To do this, the Arduino-ROS package [33] called ROSSERIAL is used. ROSSERIAL is a library which extends the abilities of the Arduino by wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character

device such as a serial port or network socket.

This way the Arduino will act as a node in the ROS architecture and the corresponding data can be accessed by any other ROS nodes as well. The configuration given in Appendix B.2 is used to setup rosserial.

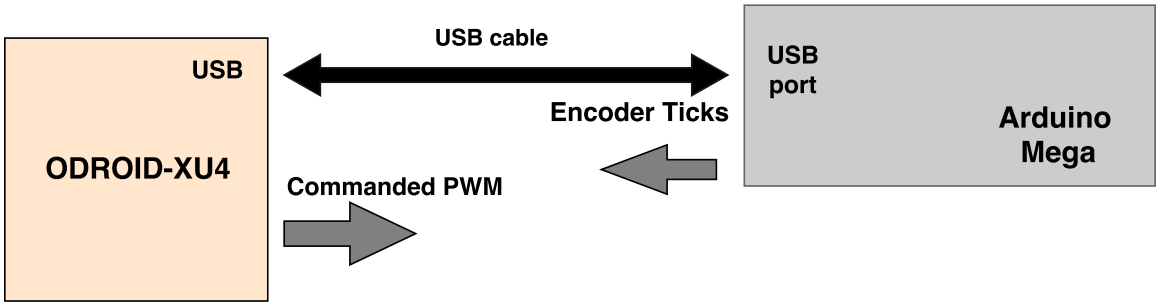


Figure 5.8: Connection between arduino and odroid that was used on ASL-Gremlin-Rover

5.2.3 Communication Setup between Arduino and Motors

The speed of the DC motor can be controlled by changing the input voltage to the motor. One way to set the speed of the motor is by sending a PWM signal to the arduino with a value of 0 being no speed and a value of 255 being the motor running at full speed. These PWM signals will be translated to a voltage signal in range 0 V - 5 V with 0 V corresponds to a PWM of 0 and 5 V corresponds to a PWM of 255.

Now, the voltage signal in range 0 V to 5 V needs to be translated to the operating voltage of the motor. In-order to have this capability, a motor-bridge is used to translate the voltage signal in range 0 V - 5 V to the motor input voltage. The motor bridge can also be used to set the direction of the wheel by reversing the motor polarity.

For the present case a dual Vnh2sp30 driver shown in Fig. 5.9 is used which is capable of driving the two motors in either direction at the same time.

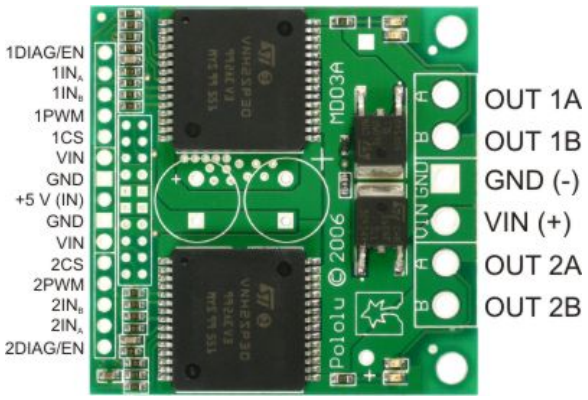


Figure 5.9: Motor driver for driving the DC motor
Source: https://www.pololu.com/product/2275

In-order to change the direction of rotation of the DC motor, the pins 1INA, 1INB for 1st motor or pins 2INA, 2INB for the second motor need to be set. By setting pin 1INA to HIGH and pin 1INB to LOW the corresponding motor will turn in a FORWARD direction. Similarly, by setting pin 1INA to LOW and pin 1INB to HIGH the corresponding motor will turn in a REVERSE direction. The final wiring diagram between Arduino, motor bridge and motors is shown in Fig. 5.10

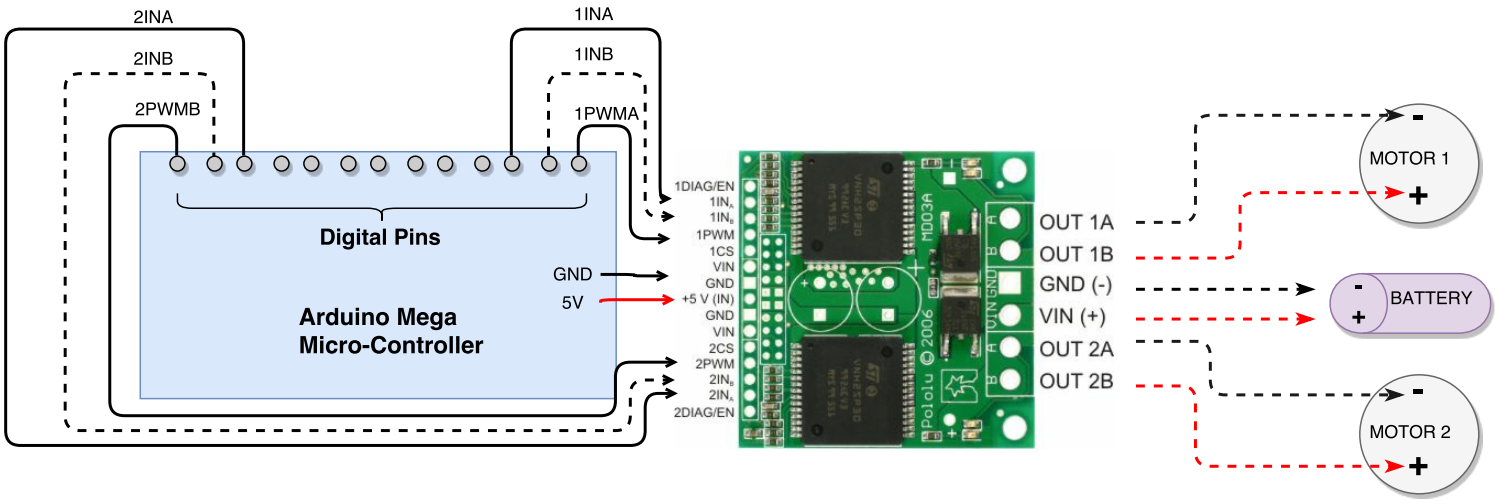


Figure 5.10: Wiring connection between arduino, motor bridge and motors

5.2.4 Communication Setup between Motor Encoders and Arduino

The quadrature encoders mounted on to the motor shaft is connected to the same Arduino that sends the PWM signals to the motors. Since the quadrature encoder gives the speed of the motor in terms of number of encoder pulses per wheel rotation, the task at hand is to increment/decrement the encoder ticks based on the speed and direction of the motor. In-order to increment/decrement the encoder ticks, the algorithm given in Appendix B.3 is used. The wiring diagram between the Arduino and a single encoder is shown in Fig. 5.11.

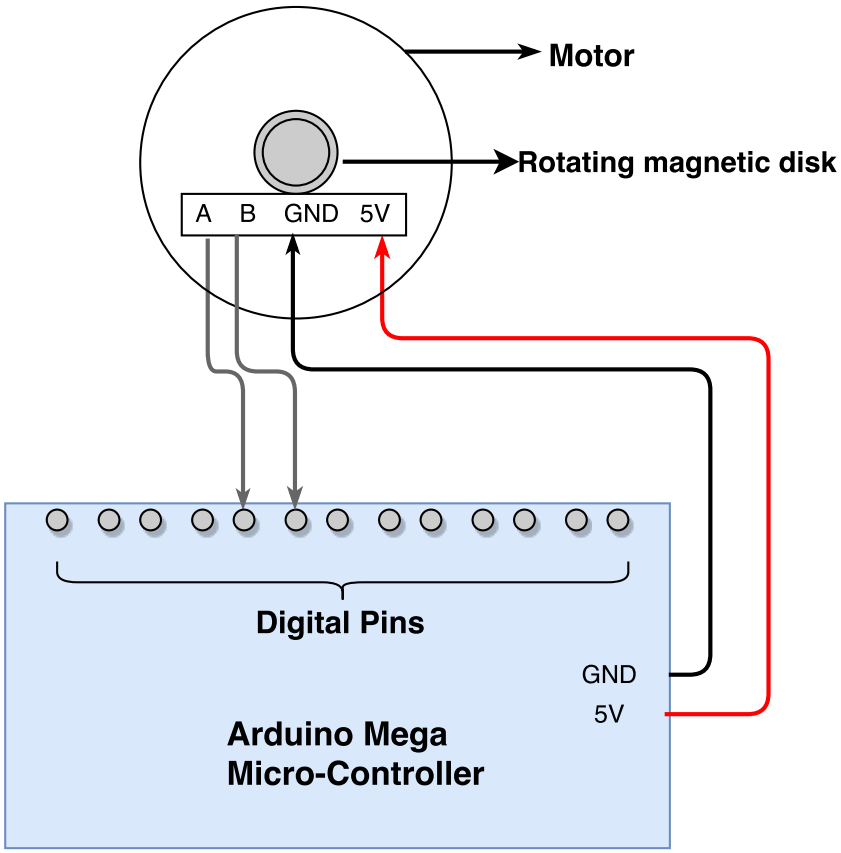


Figure 5.11: Wiring diagram of arduino and quadrature encoder

By following the above sections, the data transfer between different components on the rover is achieved and Fig. 5.12 shows the top-view of all the components on the rover.

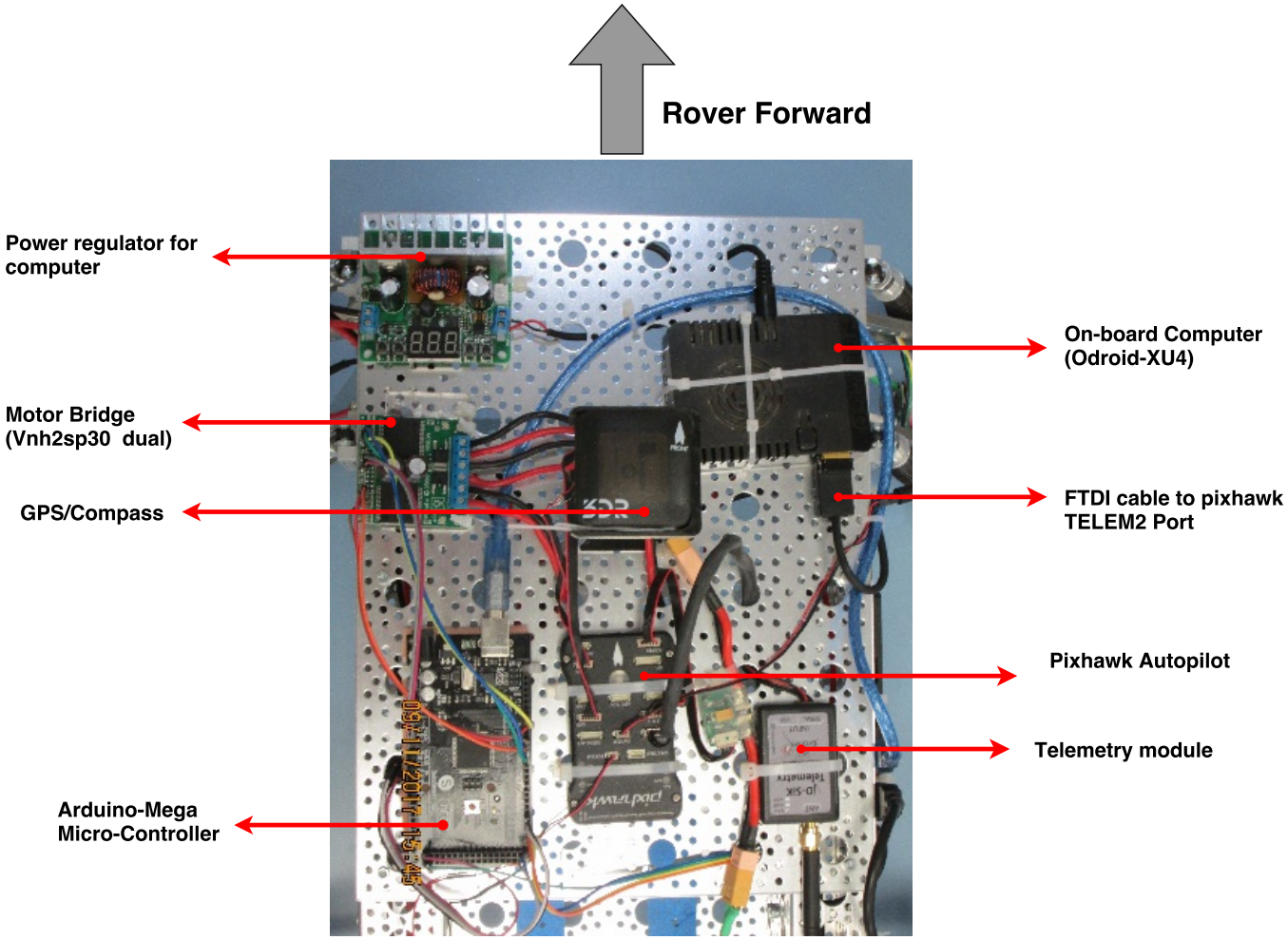


Figure 5.12: Top-view of the ASL-Gremlin-Rover with all the components

5.2.5 Data Flow Layout on the Rover

The on-board computer (Odroid) will act as a main communicating device for all other components on the rover. The sensor data from the Pixhawk is stored or processed by the Odroid and the pixhawk configuration commands are sent to

Pixhawk from Odroid. The commanded PWM signal from the controller is sent from Odroid to Arduino, which sets the required motor voltage via the motor bridge (dual Vnh2sp30). The encoder pulses from the quadrature encoders, attached to the motor shaft are sent from Arduino to Odroid for further processing. The whole layout of the data flow is shown in Fig. 5.13.

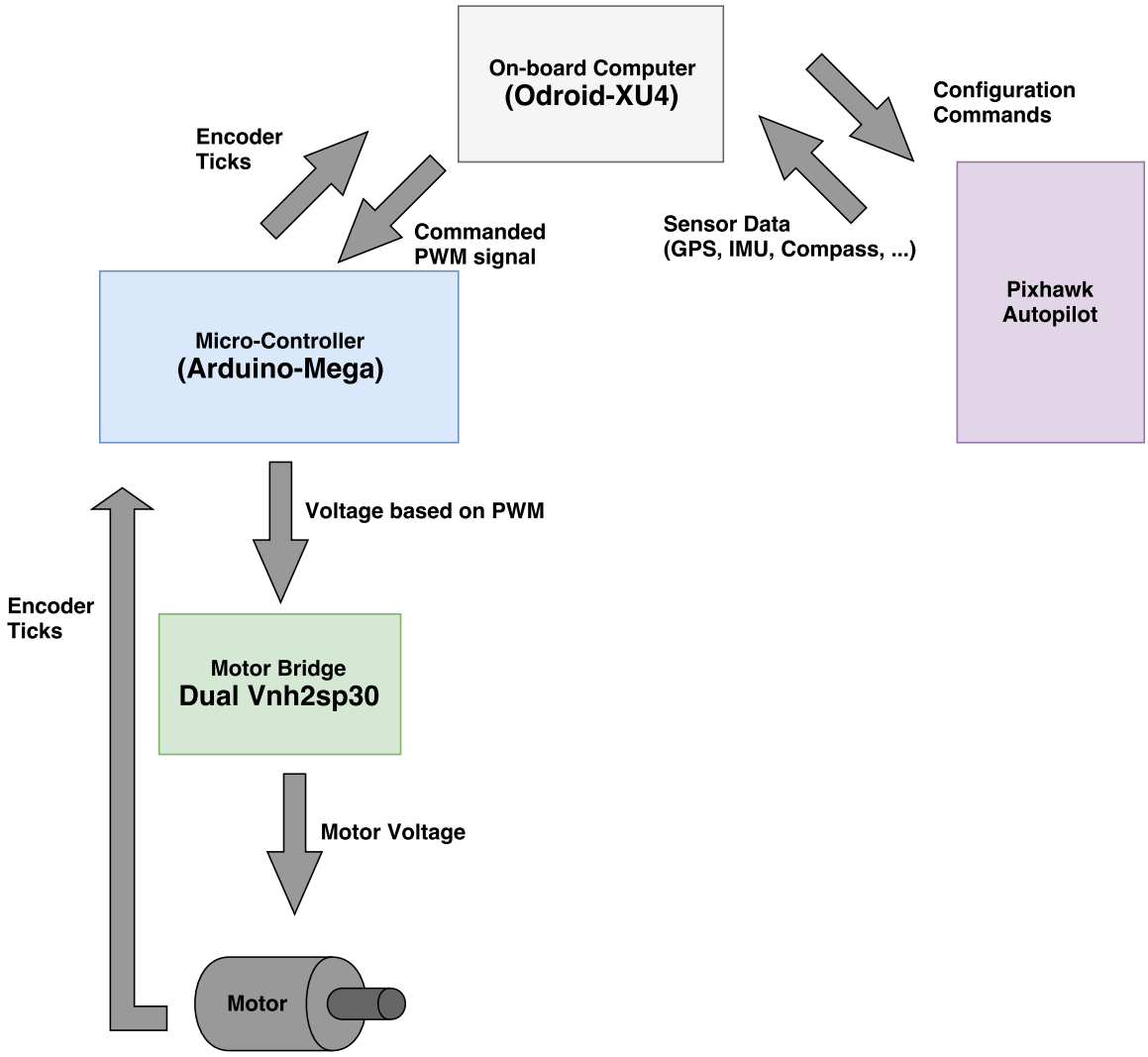


Figure 5.13: Data flow between on-board computer and different components on the Rover

5.2.6 Communication setup between Rover and Ground Station

Once the configuration/setup is completed as discussed in the earlier sections, the minimum-jerk trajectory and the backstepping controller can be tested. One way to test these methods is to write the entire trajectory generation and the controller design in C++/Python (as ROS accepts these two languages) and port it on to the on-board computer which is a time-consuming process. So, to rapidly test the proposed algorithms, a cyber-physical system based architecture is used. A cyber-physical architecture is a framework where a decentralized machine(ground station) handles all the computation and the control signals are sent to the rover based on the sensor feedback obtained from the rover. The communication between the rover and the ground-station can be handled via a network connection, Radio signals, satellite uplink etc. The advantage of this architecture is that the robot only has to handle limited amounts of computation. Another advantage is that any kind of high-sensitive information won't be lost when the robot is being operated under constrained environment.

To setup the cyber-physical system architecture, the network communication framework provided by ROS [31] is used. The reason to use the ROS network setup is that it requires minimal configuration to create the network communication. The configuration for the network communication setup is given in Appendix B.4.

A laptop[2] which runs MATLAB/SIMULINK along with the Robotics Systems ToolBox [34] from MATLAB is used as a ground station. This approach is used because the Robotics Systems Toolbox has a built-in ROS support. This gives the ability to directly use the modules that was tested based on a software simulation in SIMULINK to directly test on a real hardware by simply replacing the output signal blocks with the ROS blocks from the Robotics Systems Toolbox. The ROS network

[2] an intel i5 5^{th} gen dual core cpu with a frequency 2.3 GHz and a RAM of 4GB

configuration for the ground station is given in Appendix B.5.

Once the network setup is done, the data from the rover can be accessed directly by SIMULINK and vice versa. The final communication diagram based on the cyber-physical system architecture between the ground station and the rover is shown in Fig. 5.14.

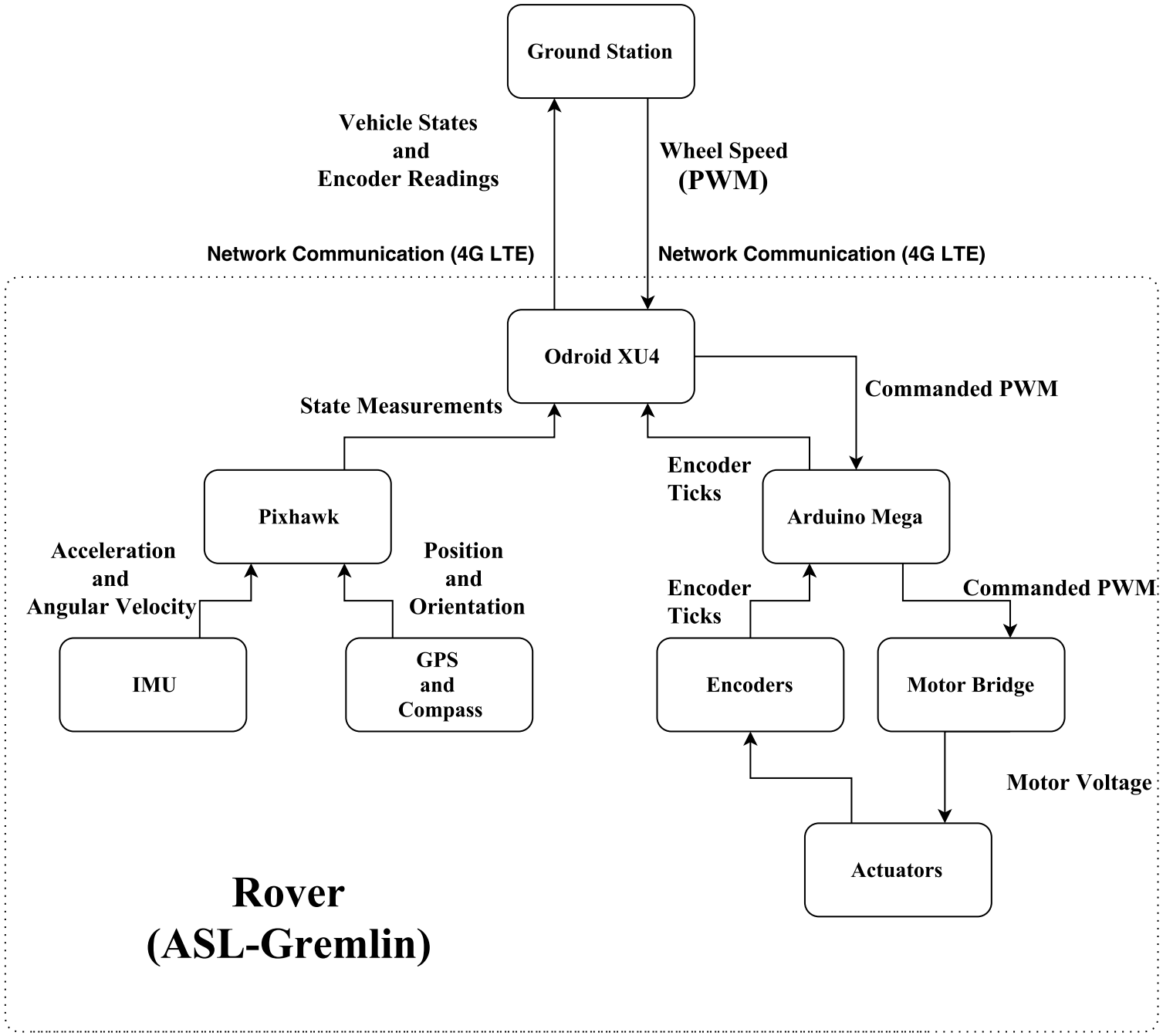


Figure 5.14: Cyber-Physical system architecture for sending and receiving data between rover and ground station

CHAPTER 6

Communication Delay Characterization and Tracking Performance Analysis

6.1 Communication Delay Characterization

There will be few pitfalls to overcome with the cyber-physical system (CPS) based architecture employed on the rover. Chief among them is the presence of network communication delays. These kind of difficulties arise due to the strength of the network being used and the capabilities of the hardware involved.

In-order to measure the network communication delay between the rover and the ground station, GPS pseudo-range based technique is used. All the data before being sent to the ground station is time-stamped which says the time-of-sent for a signal. And when a signal reaches the ground station, a time-of-arrival of signal is recorded. So the delay between the rover and the ground station is the difference between the time-of-sent and time-of-arrival of a particular signal. This method requires that both the clocks (on the ground station and on the rover) need to be in sync, which is not possible because of the time-drift in systems. So, at a fixed time-interval the local clock offset between the ground station and the rover is measured. Then, by adding or subtracting the local clock offset to the time-of-arrival of signal, an approximation of the communication delay is obtained. This procedure is shown in Fig. 6.1.

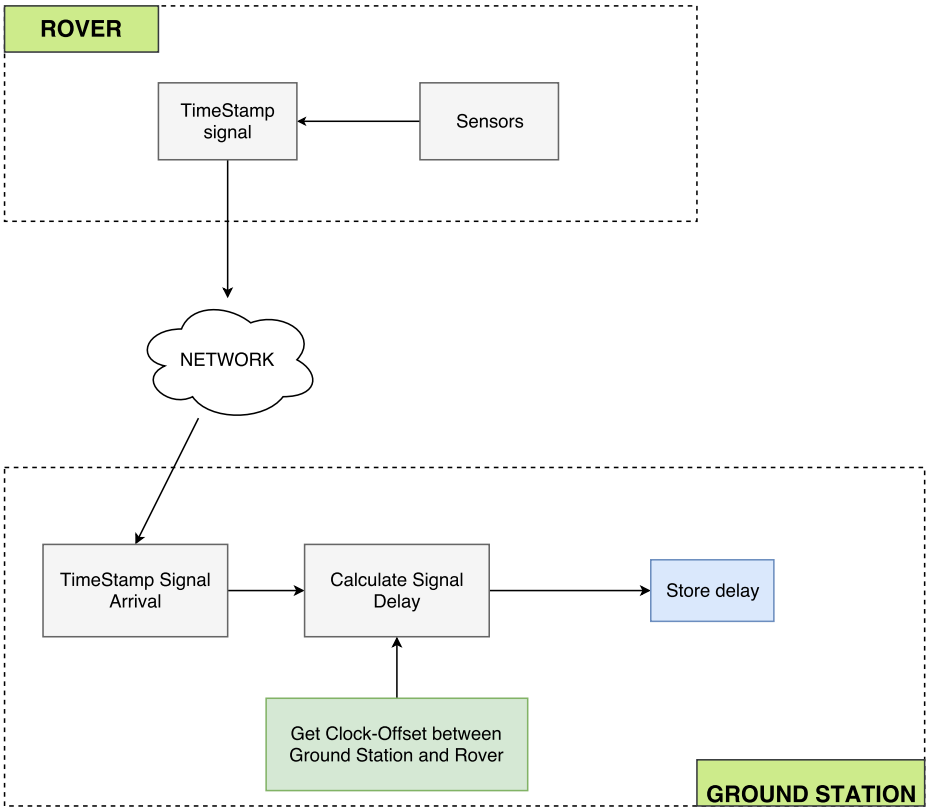
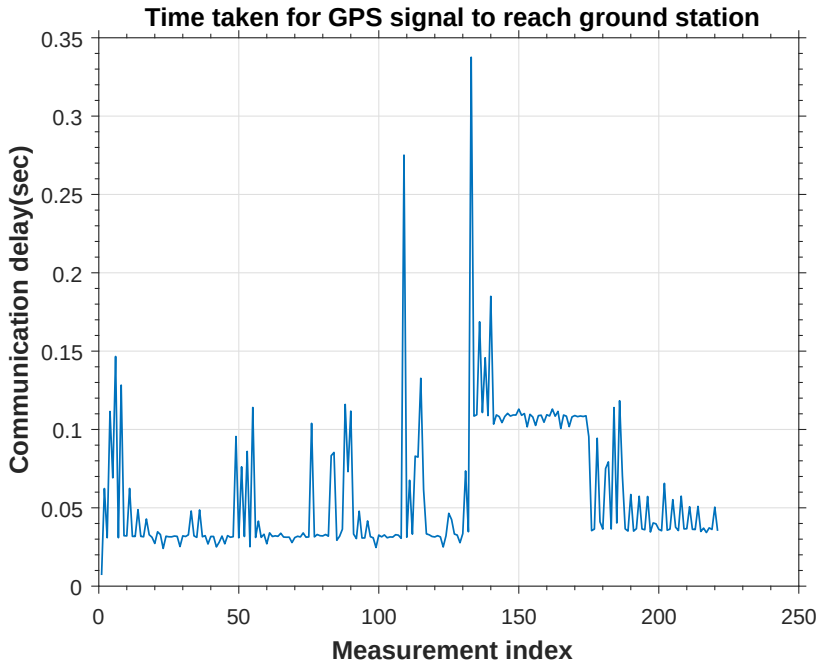
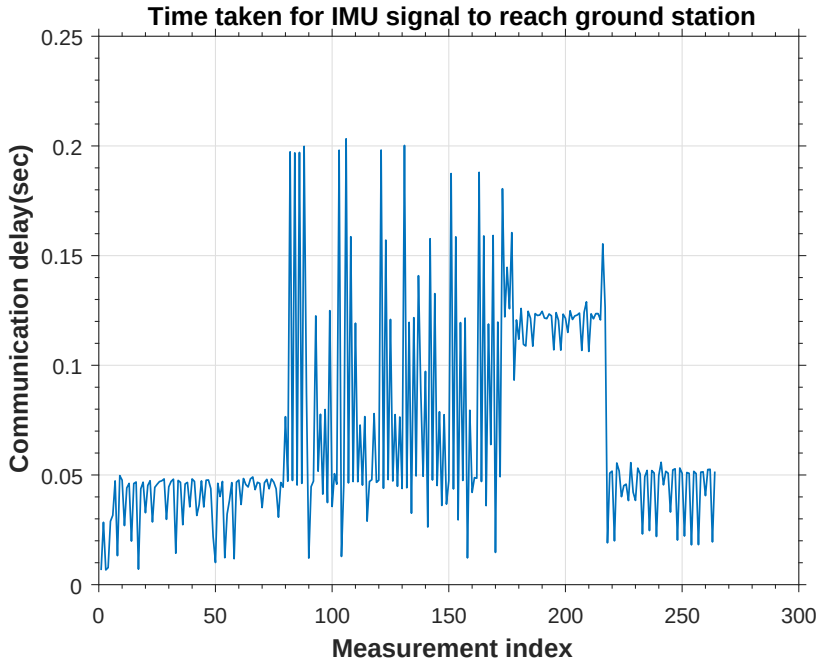


Figure 6.1: Illustration of time-delay calculation in getting the signal from rover to ground station

Fig. 6.2 shows the communication delay between the rover and the ground-station when an indoor-wifi is used with the technique shown in Fig. 6.1.

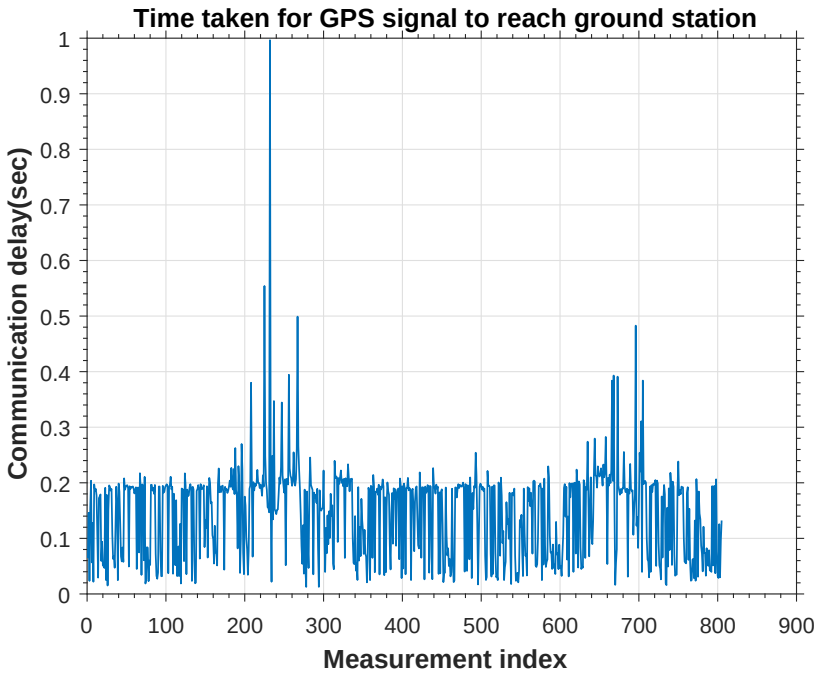


(a) Delay in getting GPS measurements

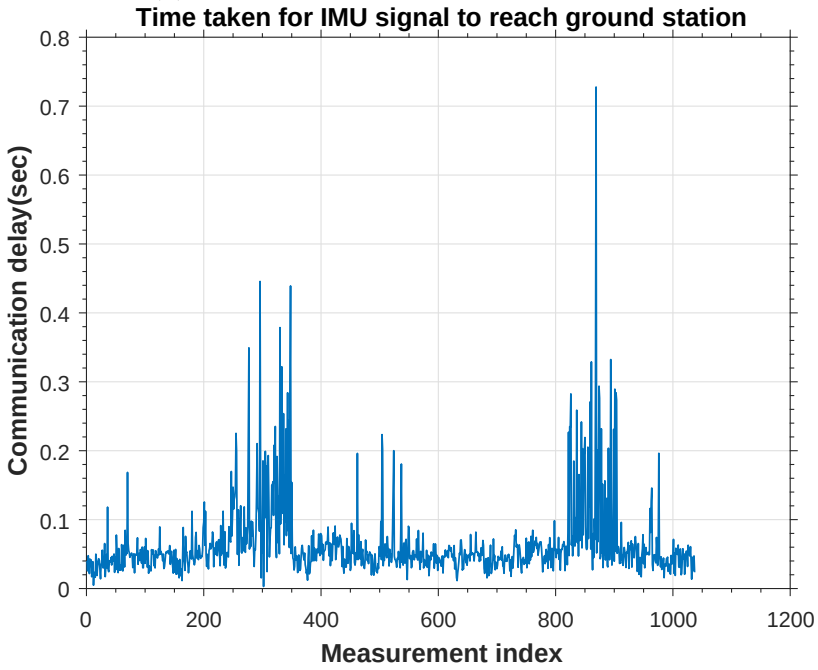


(b) Delay in getting IMU measurements

Figure 6.2: Network communication delay between the rover and ground station with indoor-wifi



(a) Delay in getting GPS measurements



(b) Delay in getting IMU measurements

Figure 6.3: Network communication delay between the rover and ground station with 4G-LTE mobile hotspot in outdoor

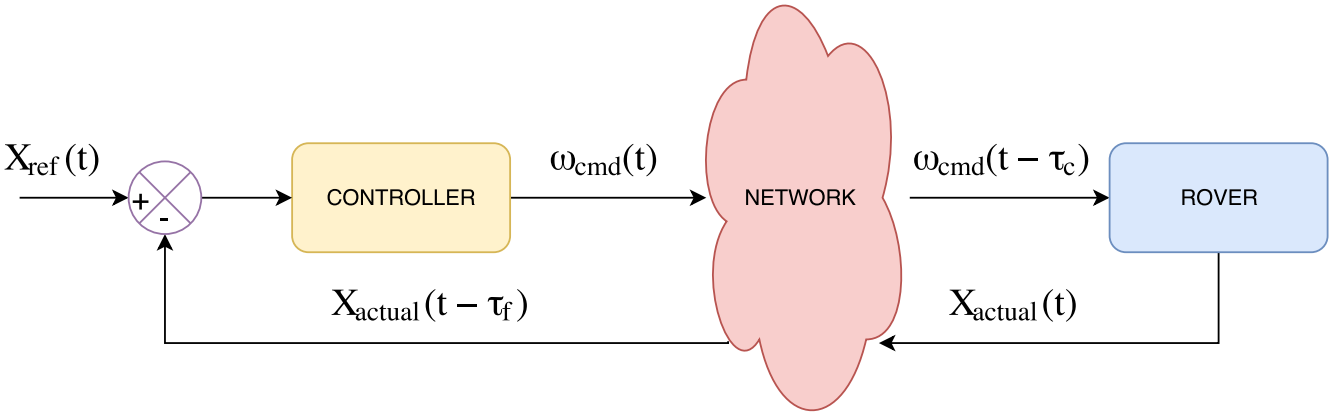


Figure 6.4: Closed loop control system with communication delay, where τ_c = control signal delay and τ_f = feedback signal delay

From Fig. 6.3, it can be said that the delays in the network can be much worse if there occurs an communication blackout. So, to observe the effect of the communication delays on the trajectory tracking performance, an artificial communication delay is generated between the controller and the rover model in simulation. A communication delay in range 0.1 sec to 0.8 sec is selected for both the control signal and feedback signal. Fig. 6.5 shows the results obtained from simulation, where 1 in the table represents rover reached the goal and 0 represents rover didn't reached the goal even though it went through some waypoints in between. As shown in Fig. 6.5, when the delay in the signal is more than 0.6 sec the performance of the system degrades due to system instability.

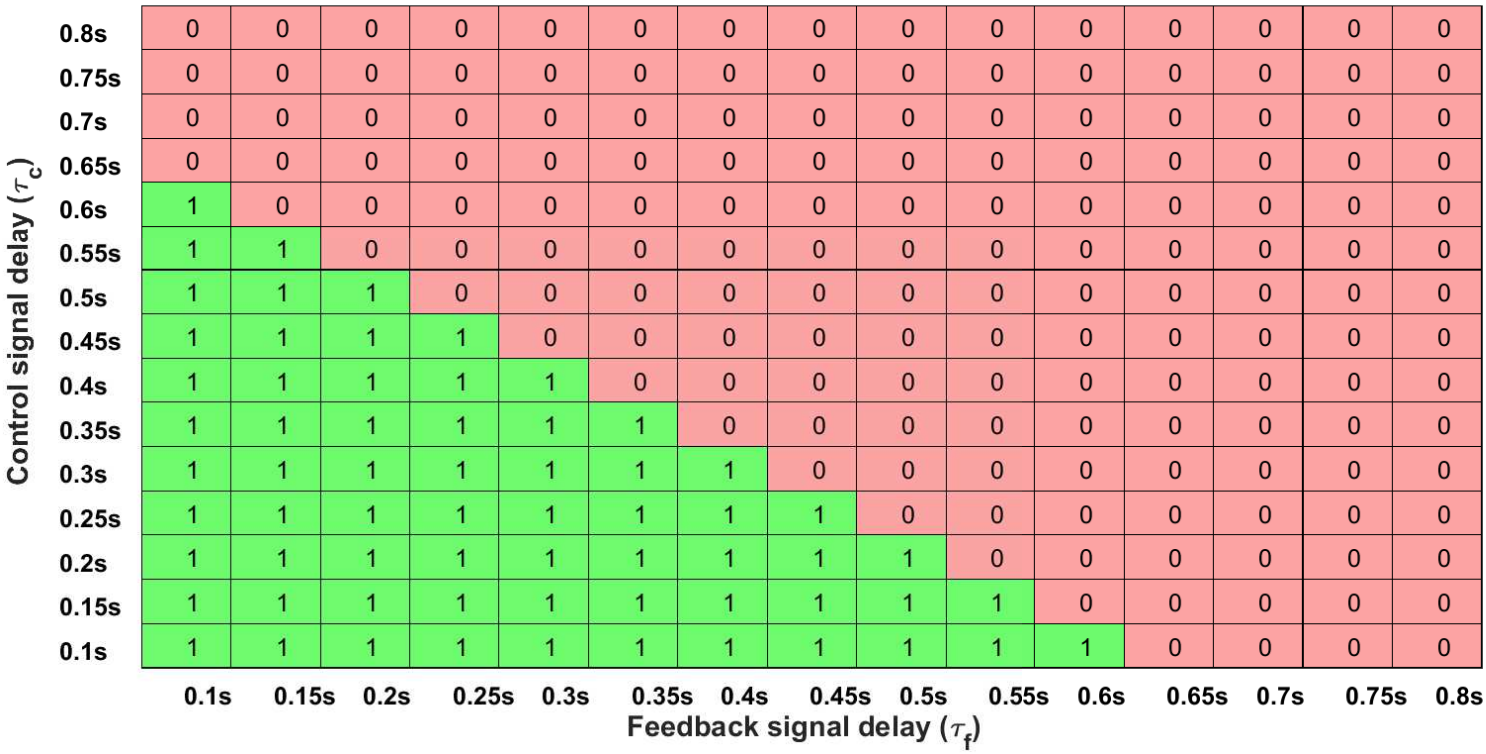


Figure 6.5: Tracking performance analysis with different combination of delays, where 1 represents vehicle reached goal and 0 represents vehicle didn't reached the goal.

CHAPTER 7

Experimental Results

This chapter discusses the experimental results by using the minimum-jerk trajectory from Chap. 3 for reference trajectory generation, backstepping controller from Chap. 4 for tracking the generated reference trajectory and the communication architecture from Sec. 5.2.6 for sending control signals to rover.

For generating the minimum-jerk trajectory a maximum acceleration of 0.1 m/sec^2 is selected and the final time for the trajectory is calculated based on that. The ground station handles the tasks like generating reference trajectory, calculating control signals and processing the encoder readings. The computer on-board the rover handles tasks like processing GPS measurements, and time-stamping sensor readings for time-delay analysis.

The IMU inside the pixhawk has an update rate of 25 Hz, GPS/Compass connected to pixhawk has an update rate of 5 Hz, and encoders attached to motors connected through arduino has an update rate based on the motor speed as hardware interrupts on micro-controller is used to get the encoder pulses. So the ground station is maintained at an update of 10 Hz if the encoders are used for rover localization and 5 Hz if the GPS readings are used for rover localization.

To get the position feedback from the rover vehicle, three different feedback methods were used mainly GPS-Compass, Encoder-Compass, and with just GPS. The GPS readings obtained from the rover has a covariance of 0.5 m^2 under clear skies and $2\ to\ 3\ m^2$ under cloudy skies. On the other hand, encoders has good accuracy for short distances but is expected to drift linearly with time for long distances especially when

the rover turns. These state-feedback techniques are discussed in-detail in Appendix C.

The following sections present the experimental results that were obtained for two types of test cases. For the results, a proximity region, or a safe-zone, of radius 2 m is considered around the waypoint. So, when the rover is within this proximity region for a particular waypoint, then trajectory is generated from the current position to next waypoint.

For the first test case, four waypoints are selected which form a square with a side length of 20 m, shown in Fig. 7.1. In the second test, the waypoints are selected to form an polygon shape with a radius of 20 m, depicted in Fig. 7.2.

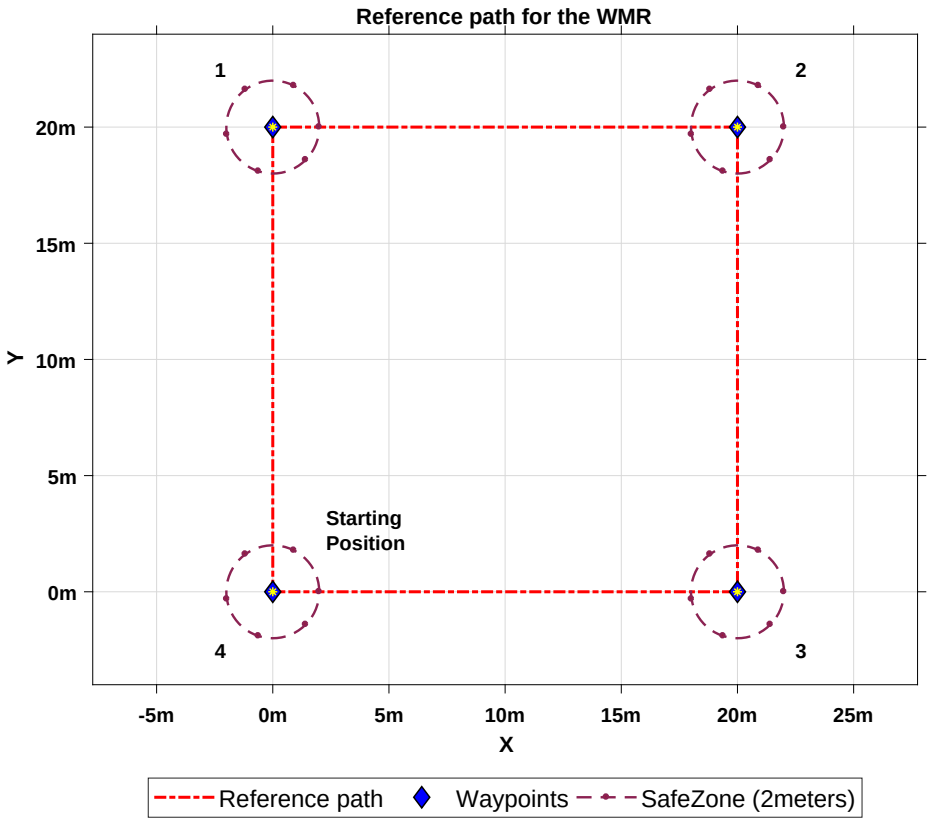


Figure 7.1: Reference path for the rover to follow for test case-1

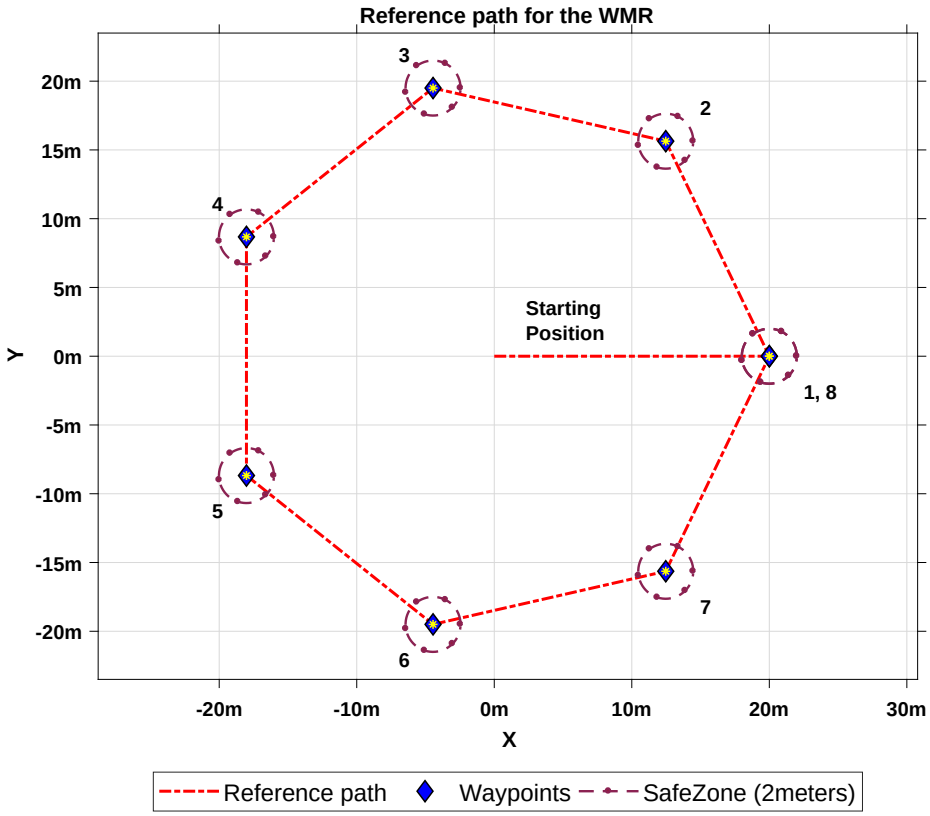


Figure 7.2: Reference path for the rover to follow for test case-2

The following subsections shows the trajectory tracking of the controller explained in Chap. 4 using the three different state-feedback techniques.

7.1 With GPS and compass

Figs. 7.3 and 7.4 shows the trajectory tracking of the rover for two test cases by using the backstepping controller from Chap. 4. It can be clearly observed that the rover follows the reference trajectory even though there are communication delays between the ground station and the rover. The nominal error between the reference trajectory and the actual rover trajectory is about 0.8 *to* 1 m can be seen from Figs. 7.5a, 7.5b. Since there is communication delay, the position feedback obtained from

the rover is not from the previous simulation time step but from some other time step given in Eq. 7.1,

$$\mathbf{E}(t) = \mathbf{X}_{actual}(t - \delta d) - \mathbf{X}_{reference}(t), \tag{7.1}$$

where δd, is the time delay of the signal to reach the ground station from the rover.

Therefore, the position error from Fig. 7.12a is much greater than what is observed from Figs. 7.3 and 7.4. At time $t = 145$ *sec* there is a sudden increase in the position error in Fig. 7.5b. This occurs because the position measurement from the previous time-step ($\mathbf{X}(t - \delta d)$) has arrived at the current time instant because of the communication delay.

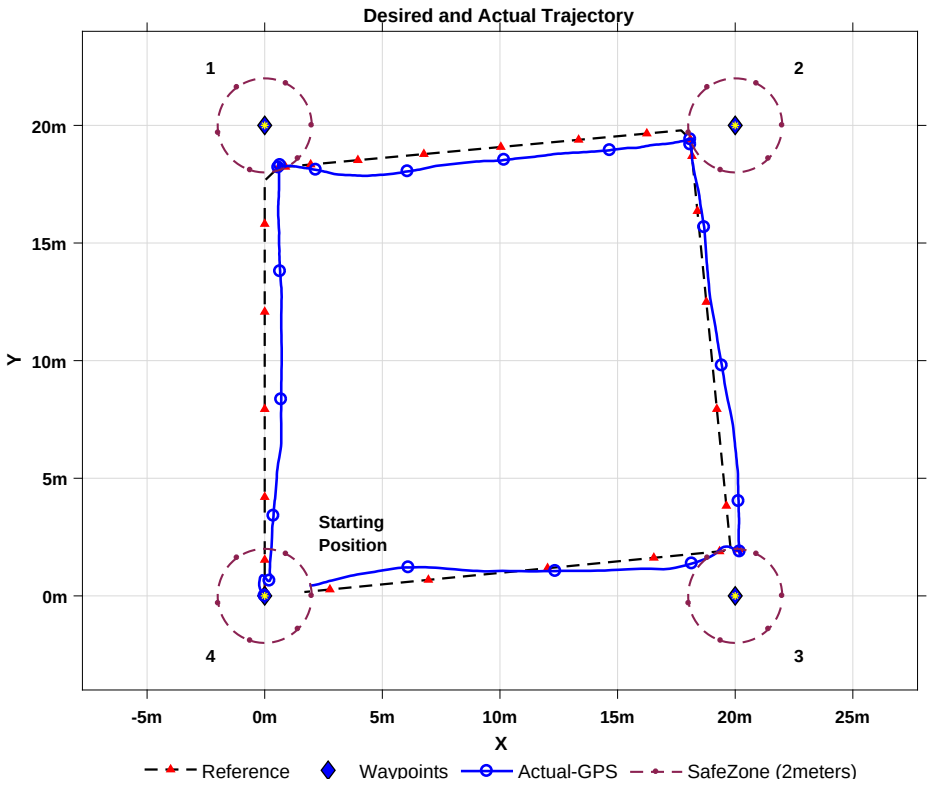


Figure 7.3: Reference and actual path of the rover for test case-1 with GPS and compass for state-feedback

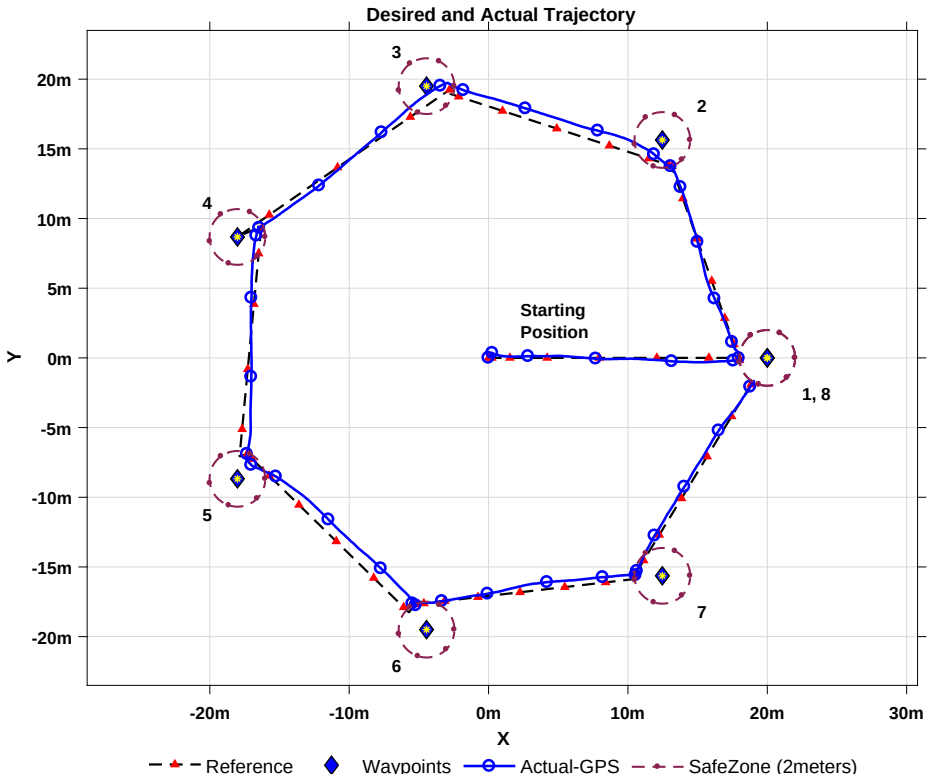
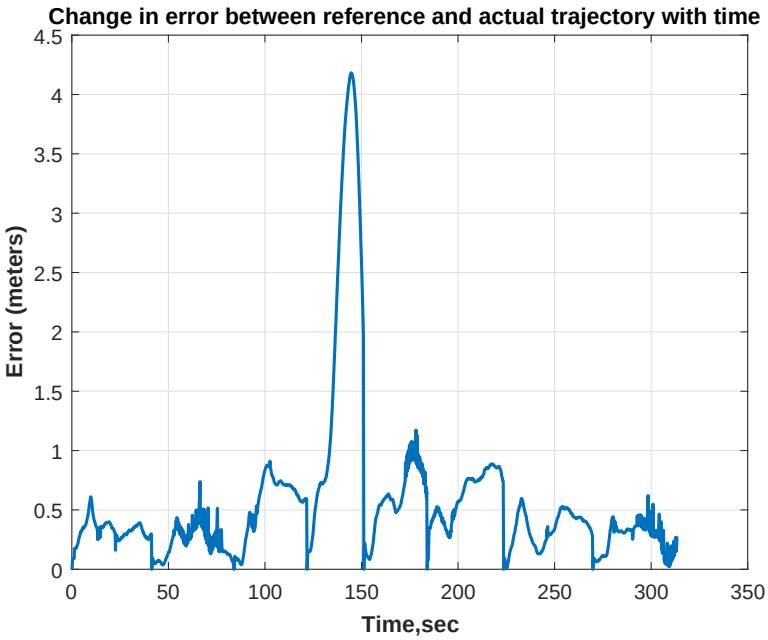


Figure 7.4: Reference and actual path of the rover for test case-2 with GPS and compass for state-feedback

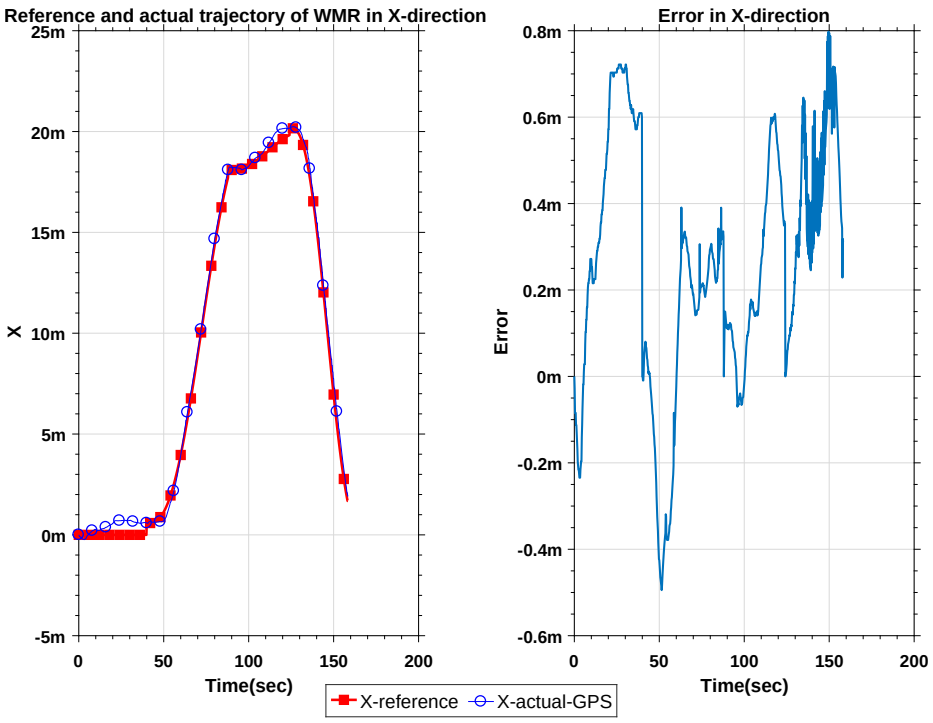
The trajectory tracking of the rover in x, y direction is shown in Figs. 7.6, 7.7. As the trajectory for the next waypoint is created when the rover reaches the proximity region of the current waypoint. At the waypoint transition, the error in position decreases and the error in the heading angle increases. The decrease in position error at the waypoint transition is because the trajectory for the next waypoint is generated from the current position of the rover. So, for the initial time instant of the trajectory generation, the position error is smaller. The increase in heading angle error around the waypoint transition is illustrated in Fig. 7.8.

(a) Test case-1

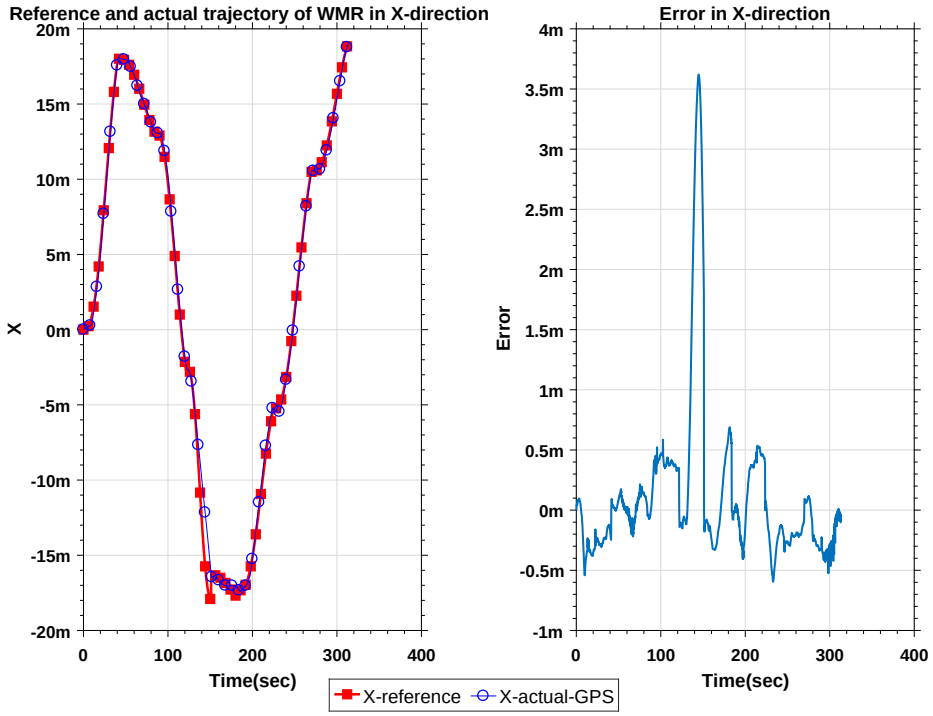


(b) Test case-2

Figure 7.5: Error in position between reference and actual trajectory of the rover with GPS and compass for state-feedback

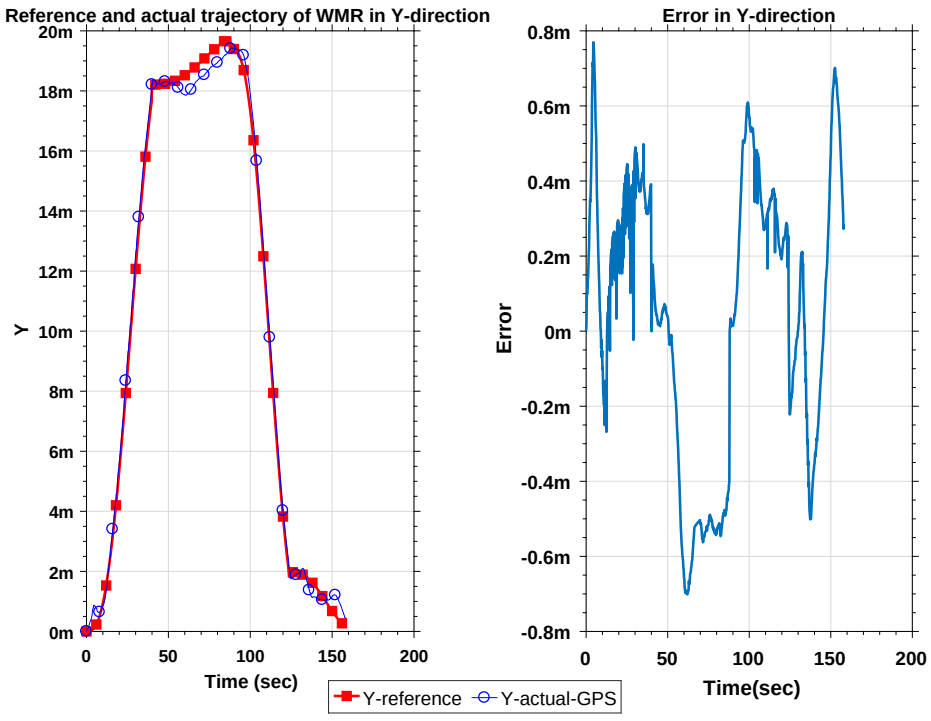


(a) Test case-1

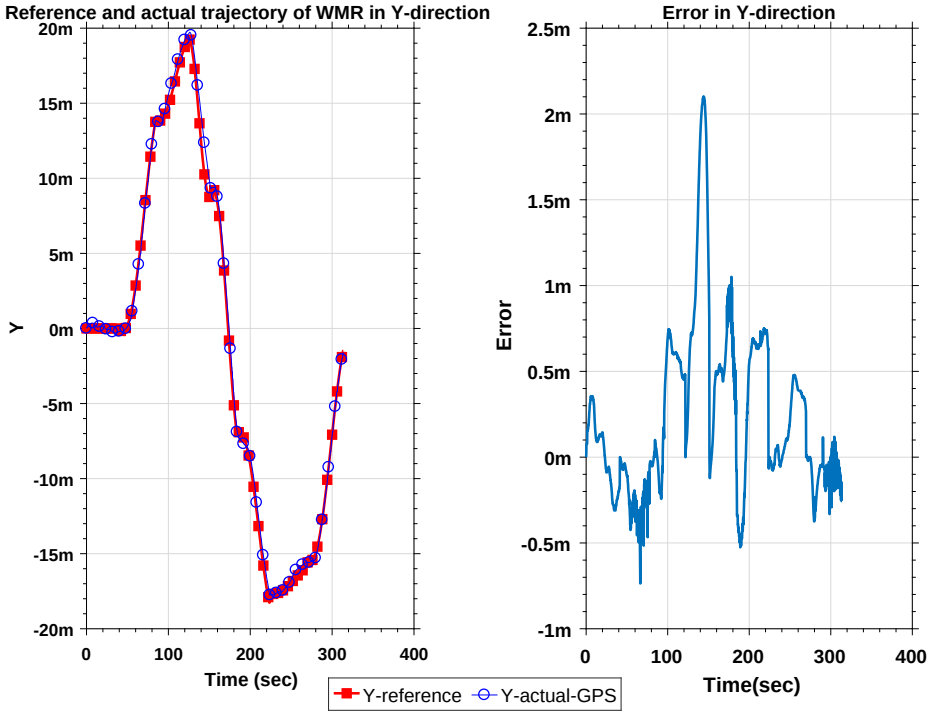


(b) Test case-2

Figure 7.6: Reference and actual trajectory of the rover in X-direction with GPS and compass for state-feedback



(a) Test case-1

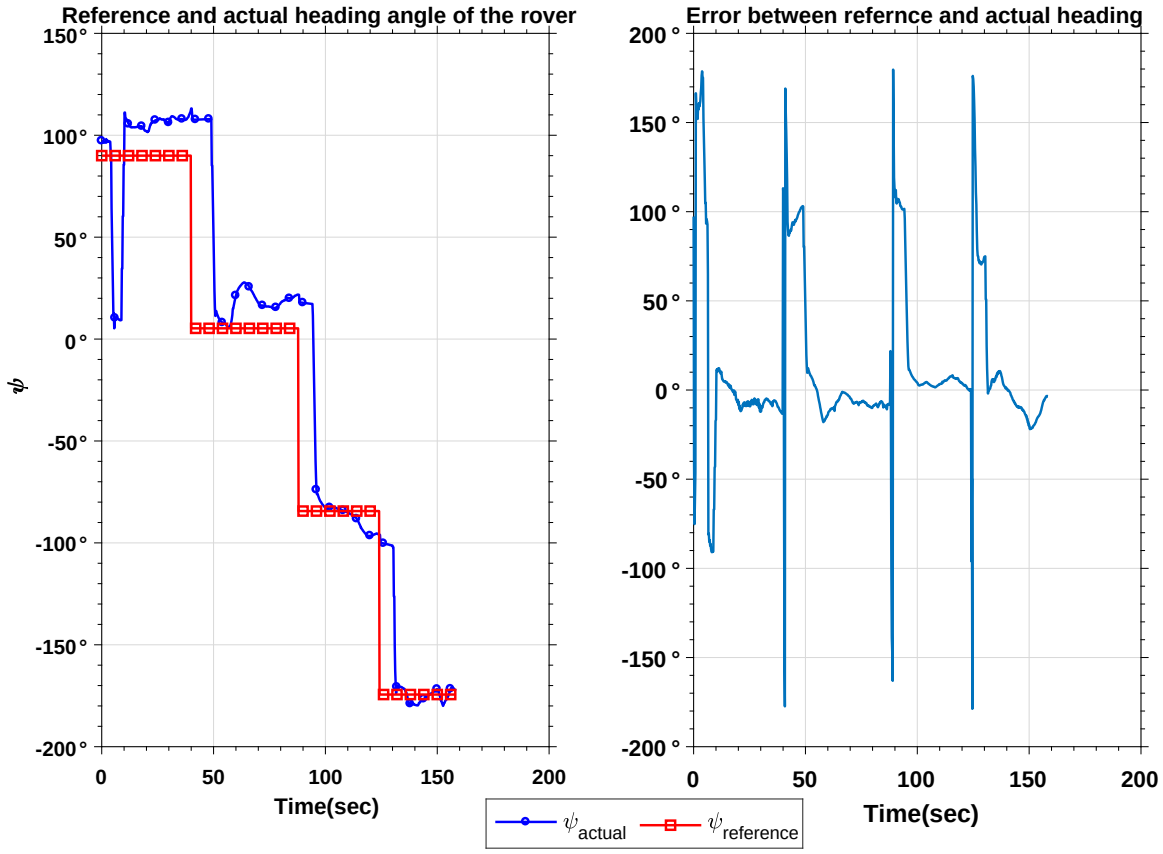


(b) Test case-2

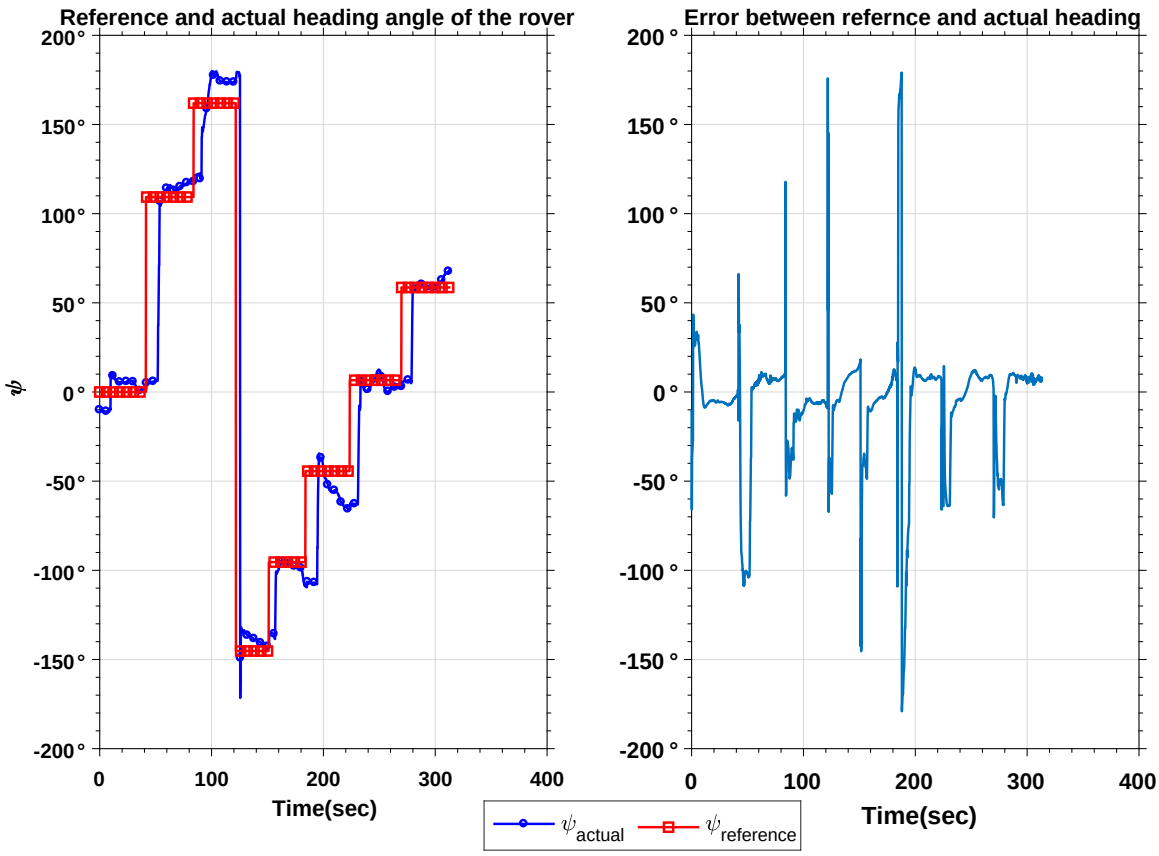
Figure 7.7: Reference and actual trajectory of the rover in Y-direction with GPS and compass for state-feedback

Because of the communication delay at time $t = 145\ sec$, there is a sudden increase in position error as the signal from some other elapsed time-step reached at the current time-instant.

The error in heading angle increases around the waypoint transition point seen in Fig. 7.8. In-order to reduce that error, the proposed controller automatically commands necessary angular velocities to the wheels such that the rover aligns itself towards the required heading direction. The commanded angular velocities by the controller around the waypoint transition to align the rover is shown in Figs. 7.9 and 7.10.



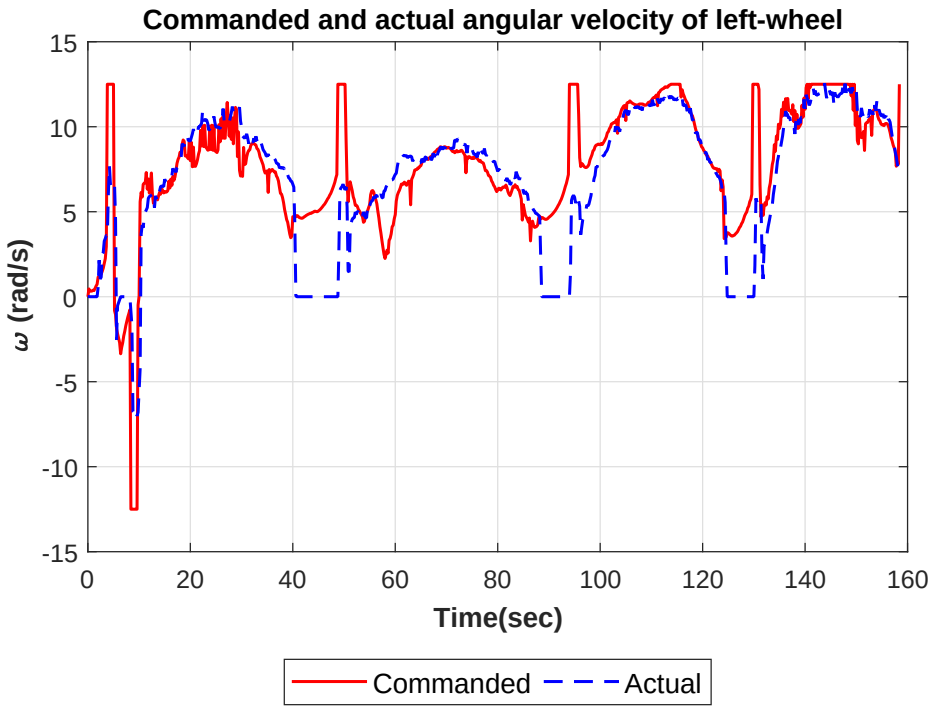
(a) Test case-1



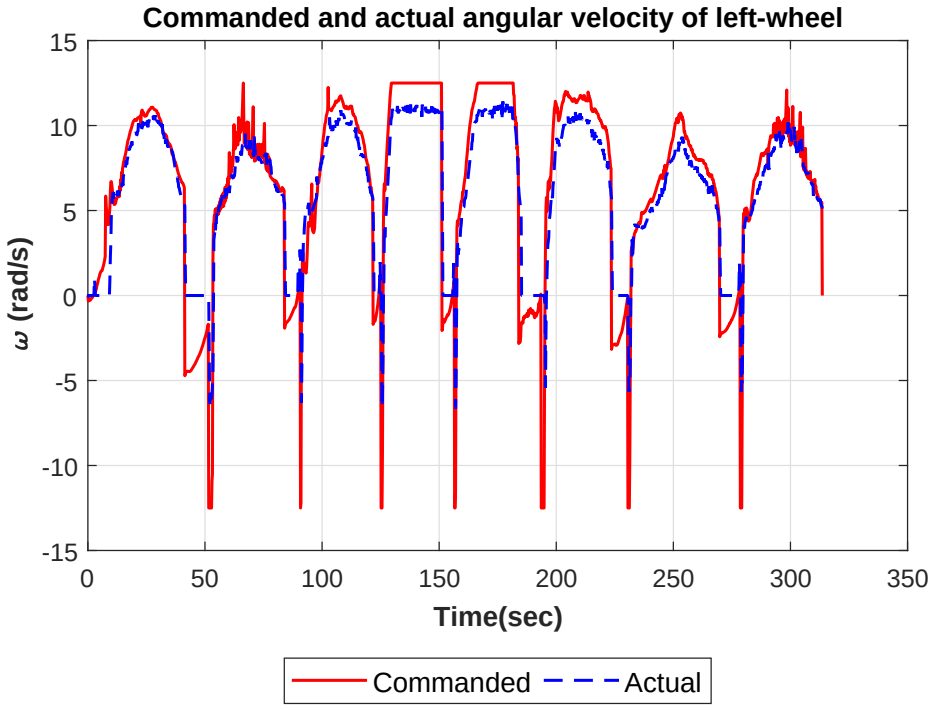
(b) Test case-2

Figure 7.8: Reference and actual heading angle of the rover with GPS and compass for state-feedback

The actual angular velocity of the vehicle is calculated by using the encoders and Eq. C.8. As the minimum jerk trajectory from Chap. 3 is used to create the reference trajectory, the maximum velocity in reference trajectory occurs at time $\frac{t_{final}}{2}$. This can be seen from Figs. 7.9a, 7.10a, where the angular velocity is maximum at time $\frac{t_{final}}{2}$ for a given trajectory segment. It can be seen that the actual angular velocity of the rover is approximately equal to the commanded angular velocity by the controller, which enables the rover to track the reference trajectory.

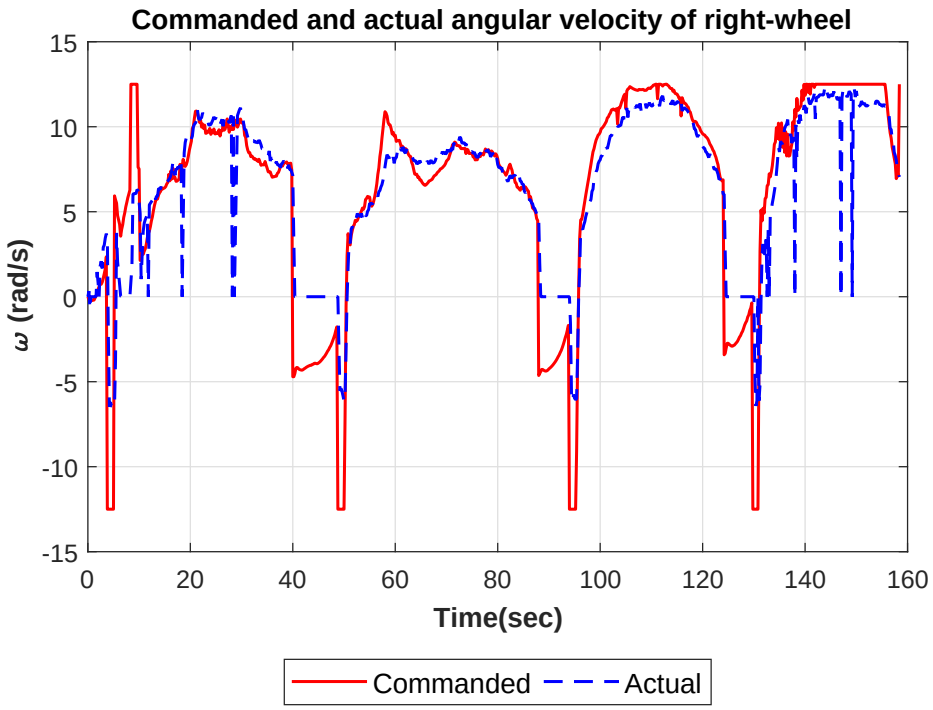


(a) Test case-1

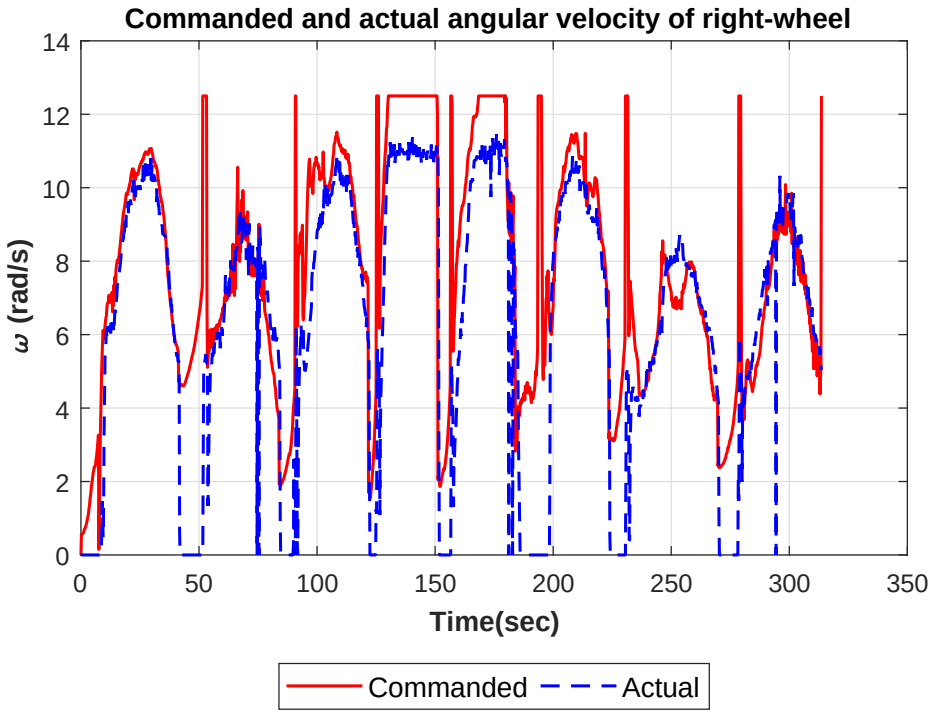


(b) Test case-2

Figure 7.9: Commanded and actual angular velocity (rad/s) of left-wheel GPS and compass for state-feedback



(a) Test case-1



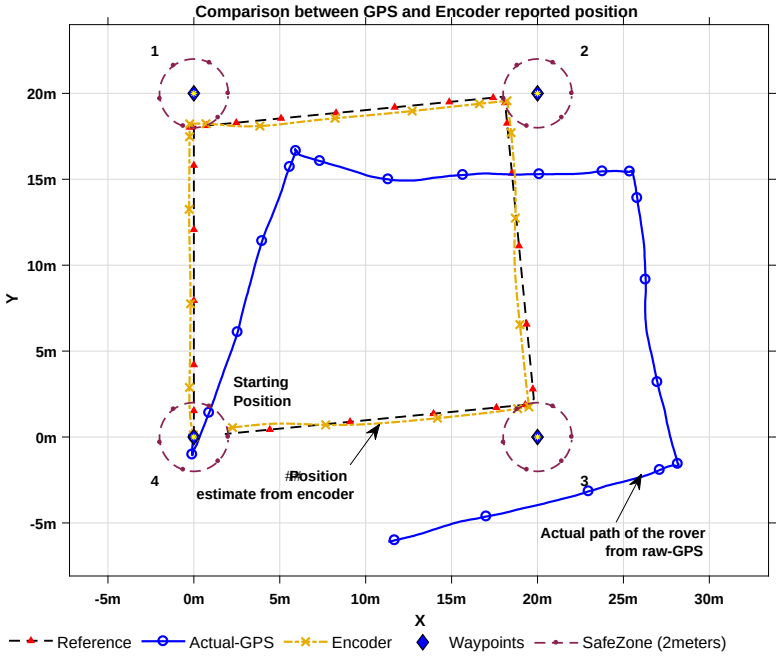
(b) Test case-2

Figure 7.10: Commanded and actual angular velocity (rad/s) of right-wheel GPS and compass for state-feedback

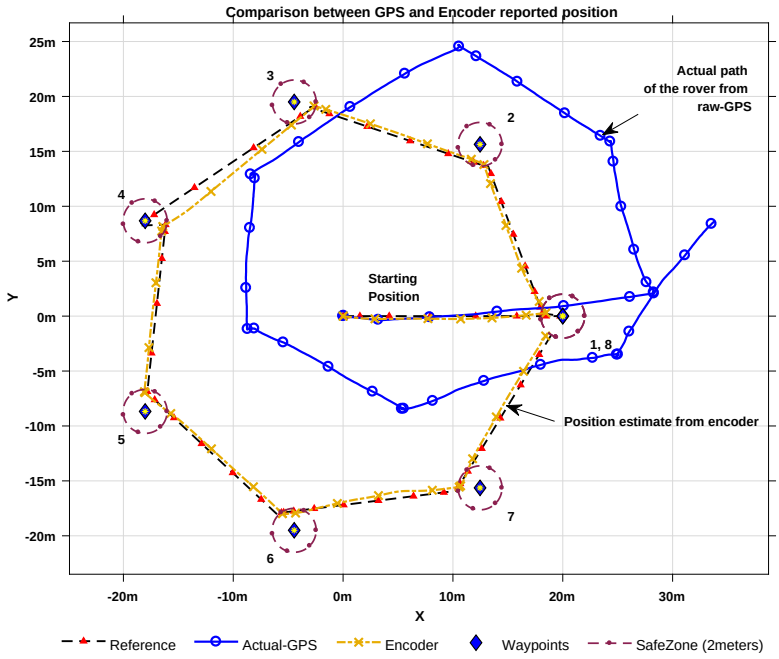
7.2 With Encoder and compass

This section presents the results obtained for the two cases shown in Fig. 7.1 and 7.2, with position feedback through the encoders (given in Appendix C) and the heading feedback from the compass. The encoder based position feedback is obtained by assuming that there is no wheel slippage, wheels are rigid.

Fig. 7.11 shows the trajectory tracking of the rover for both the test cases by using the encoders and compass for state-feedback. It also shows the drift in encoder readings by comparing the encoder reported position to the actual GPS readings. Even though the encoder says that the rover reached the goal position and tracked the reference trajectory, the GPS readings however, shows the actual path that the rover tracked. This made the rover reach a location which is 6 to 8 m away from the goal. The rover however, ended up making a square trajectory due to the heading reference gathered from the compass. This allowed the rover to correct it's heading angle despite the growing position errors.

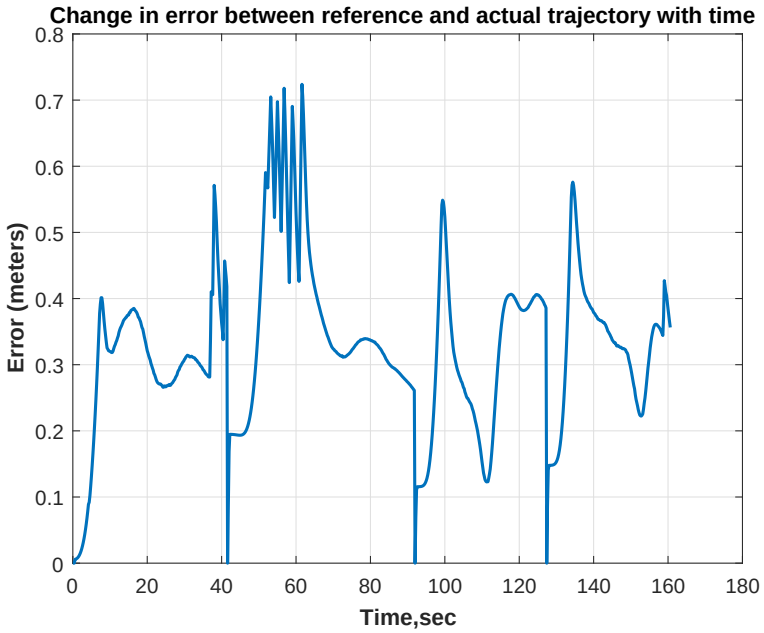


(a) Test case-1

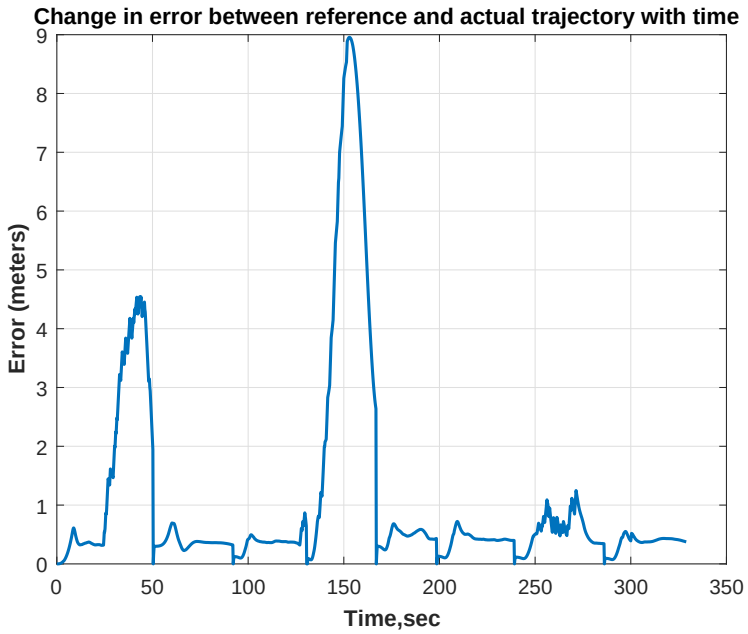


(b) Test case-2

Figure 7.11: Comparison between GPS and Encoder reported position with encoder and compass for state-feedback



(a) Test case-1

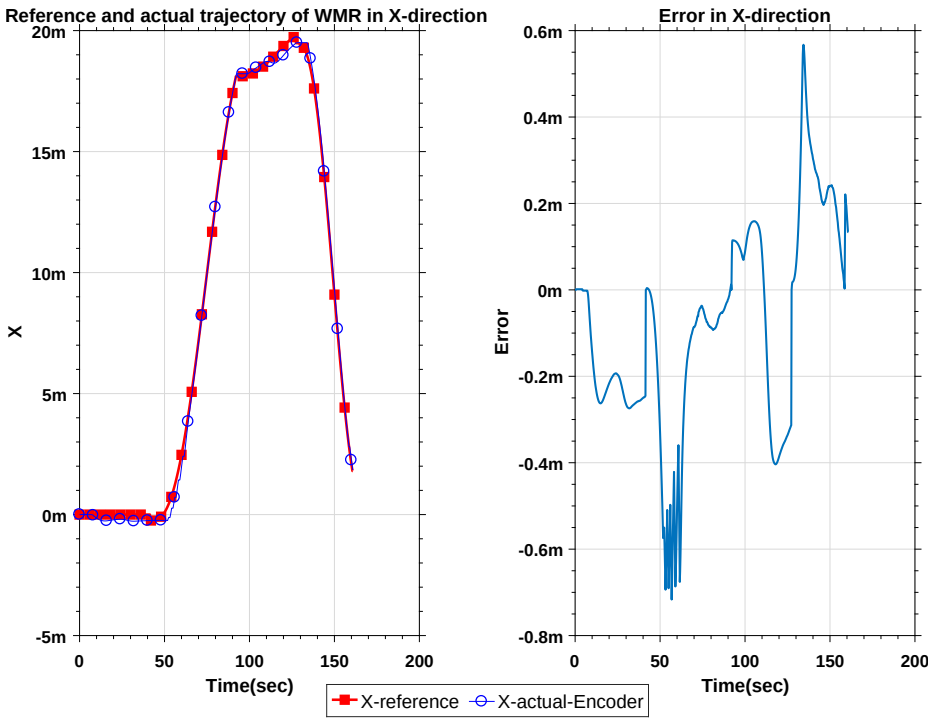


(b) Test case-2

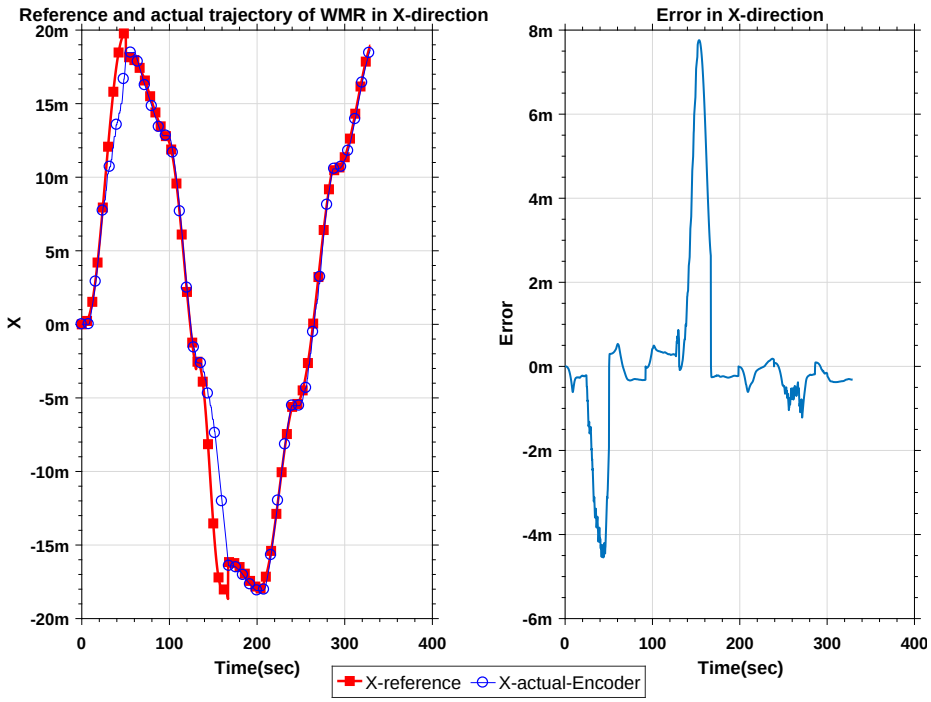
Figure 7.12: Error in position between reference and actual trajectory of the rover with encoder and compass for state-feedback

The trajectory tracking of the rover in x, y direction is shown in Fig. 7.13, 7.14. There is an increase in position error at some instants because of the communication delays. As the error in position increases in Fig. 7.13a, the controller tries to minimize the error by increasing the angular velocity of the wheels. At time $t = 50$ *sec* from Fig. 7.13a there is an increase in position error. So, as the position error increases at $t = 50$ *sec*, the commanded angular velocity depicted in Figs. 7.16a and 7.17a also increased in-order to minimize the error.

As the error in heading angle increases around the waypoint transition, the controller commands the necessary angular velocity in-order to align the rover towards the required heading angle. The commanded angular velocity by the controller around the waypoint transition can be seen clearly in Figs. 7.16 and 7.17.

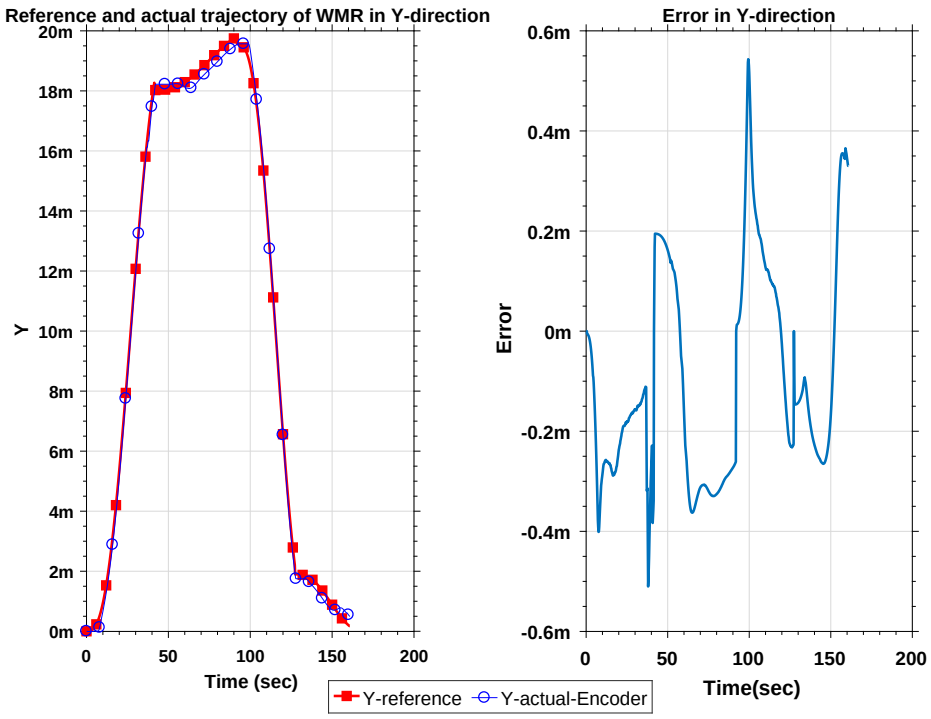


(a) Test case-1

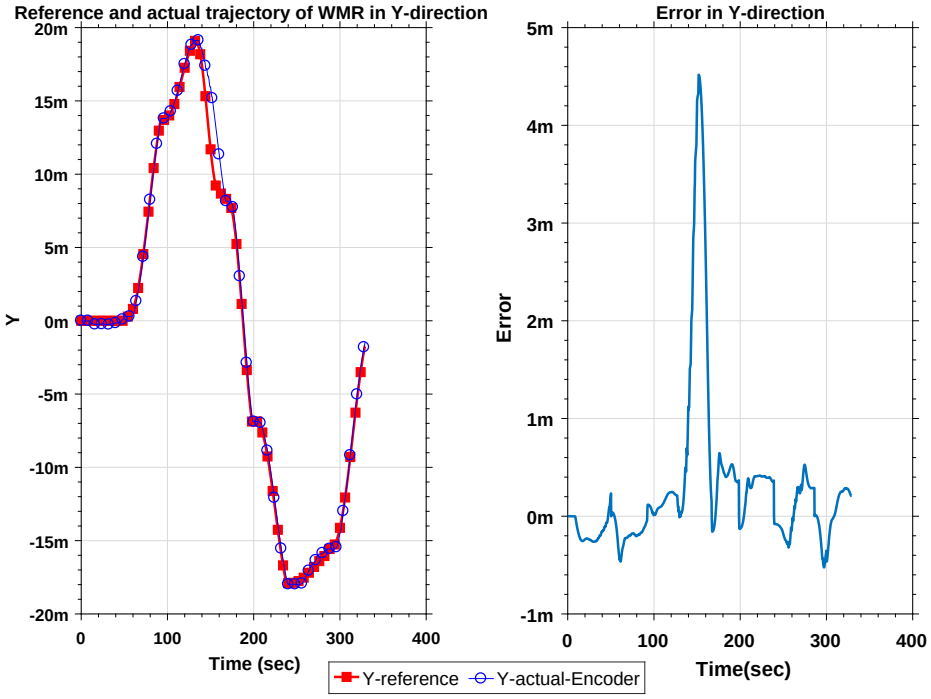


(b) Test case-2

Figure 7.13: Reference and actual trajectory of the rover in X-direction with encoder and compass for state-feedback

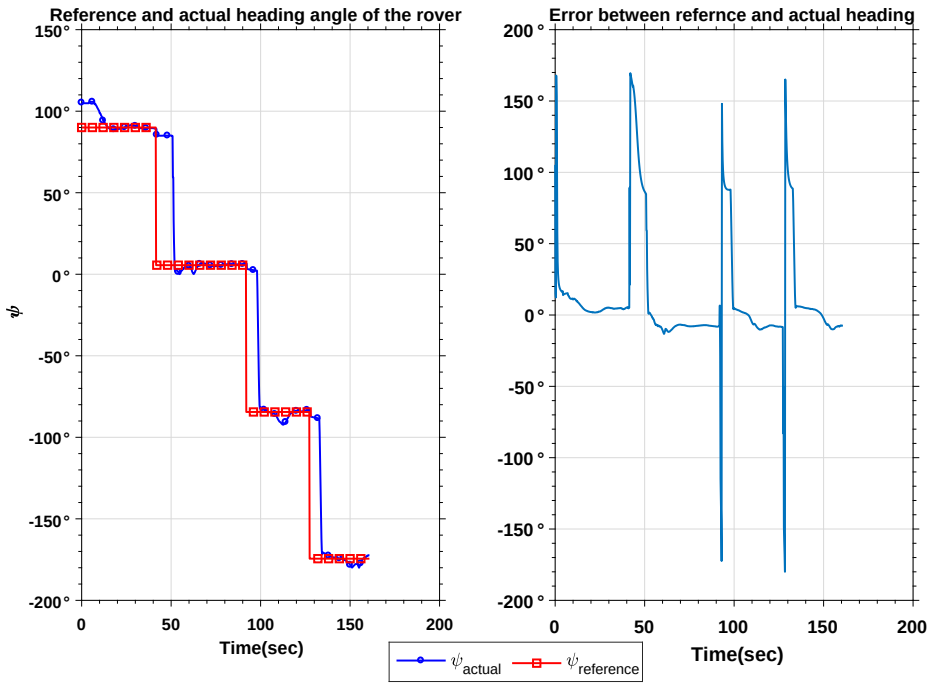


(a) Test case-1

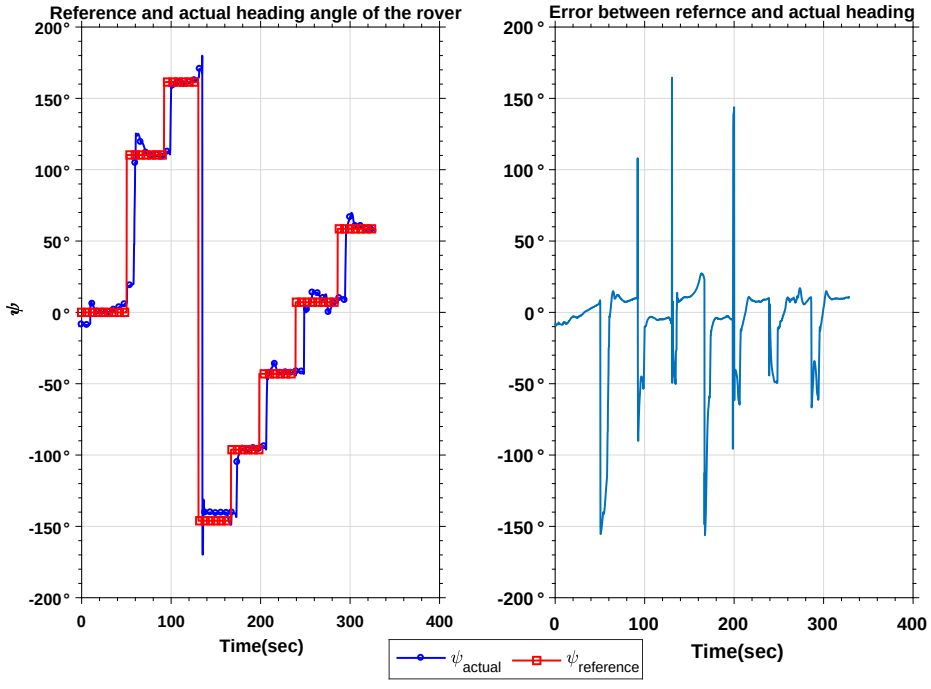


(b) Test case-2

Figure 7.14: Reference and actual trajectory of the rover in Y-direction with encoder and compass for state-feedback

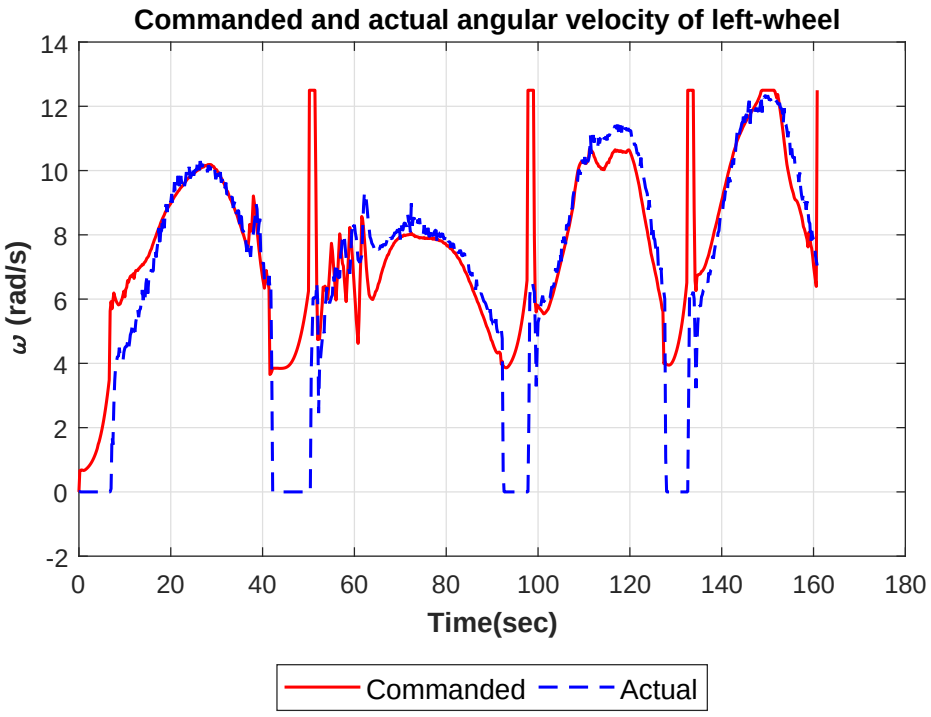


(a) Test case-1

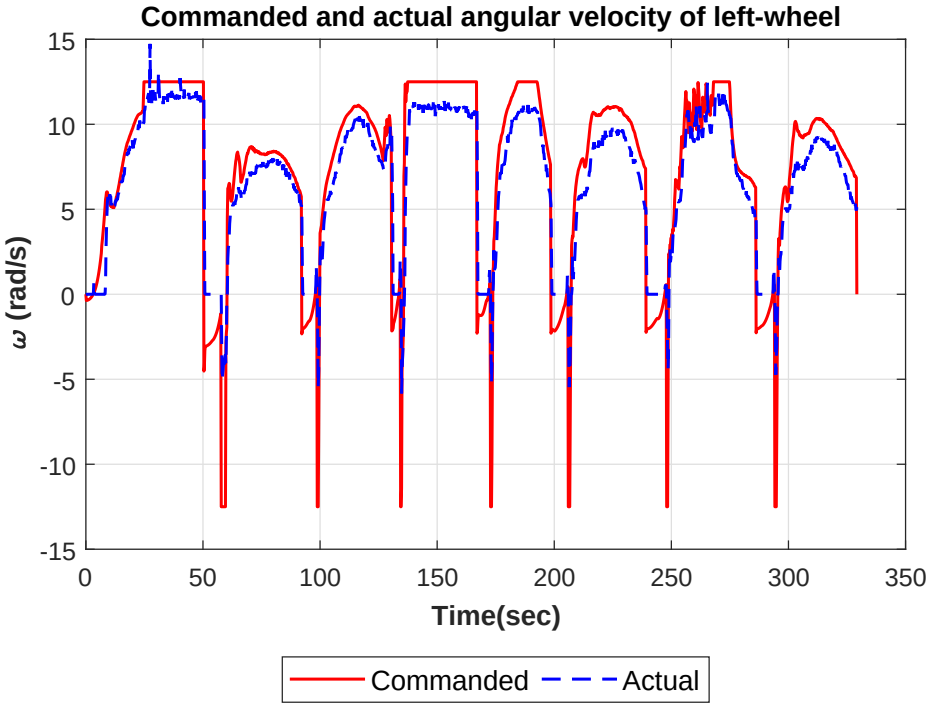


(b) Test case-2

Figure 7.15: Reference and actual heading angle of the rover with encoder and compass for state-feedback

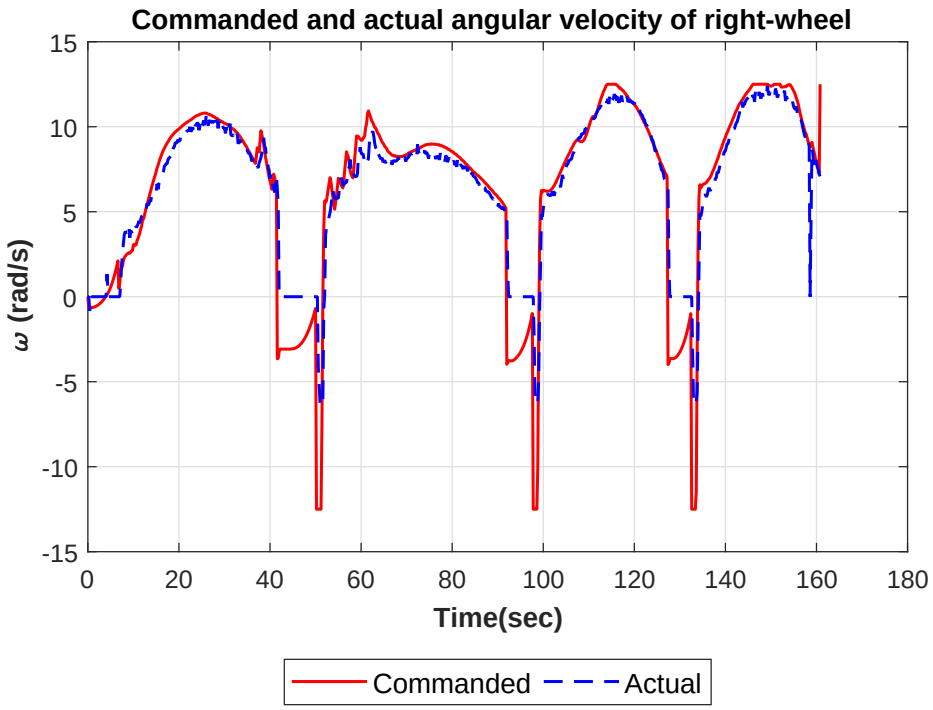


(a) Test case-1

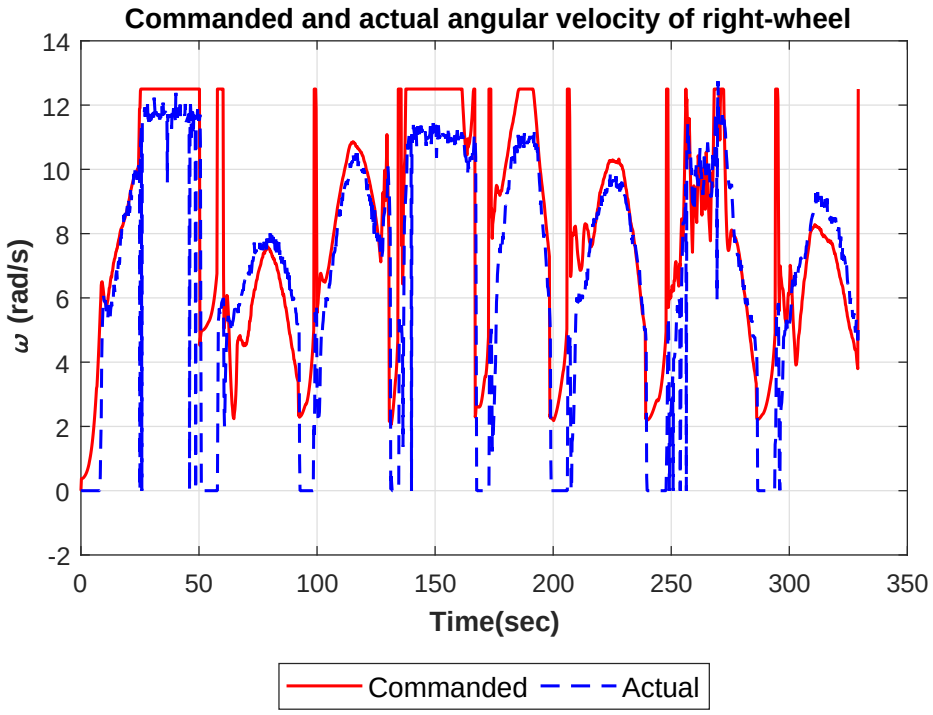


(b) Test case-2

Figure 7.16: Commanded and actual angular velocity (rad/s) of left-wheel with encoder and compass for state-feedback



(a) Test case-1



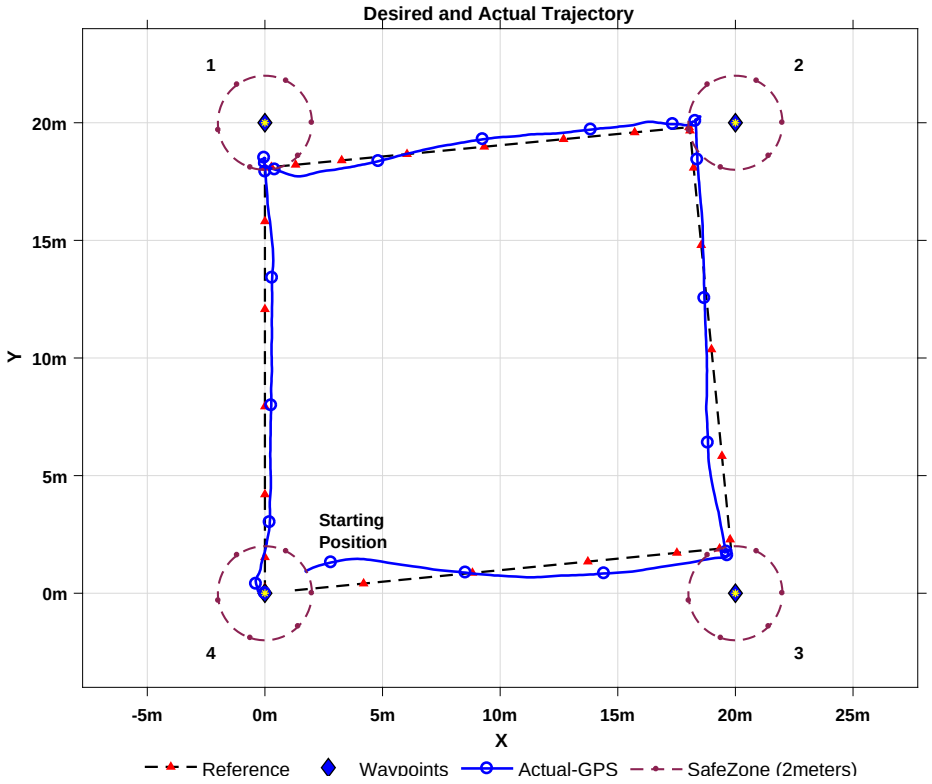
(b) Test case-2

Figure 7.17: Commanded and actual angular velocity (rad/s) of right-wheel with encoder and compass for state-feedback

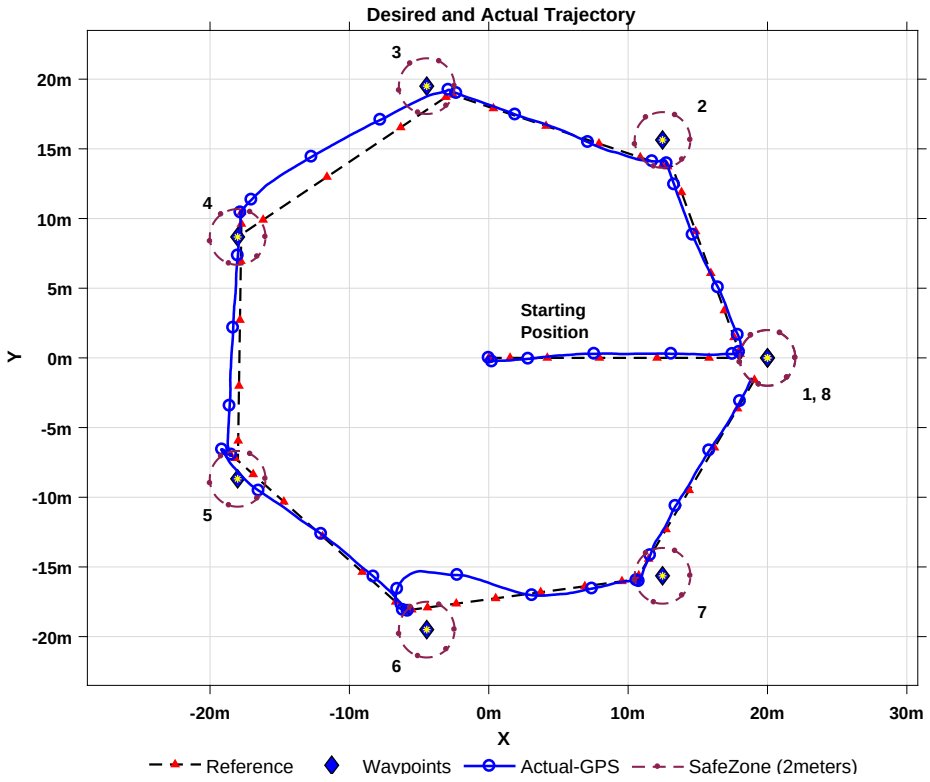
7.3 With only GPS

For this case, GPS measurements are converted to the ENU frame using the same method discussed in Appendix C. To get the heading angle feedback, the tangent angle between the previous GPS measurement and the current GPS measurement is taken, which gives the angle in ENU co-ordinates.

From Fig. 7.18, it can be observed that the controller tracks the reference trajectory closely. As there is a presence of network communication delays, asynchronous data processing for error calculation is observed and given by Eq. 7.1. So, the nominal position error is about 1.8 m shown in Fig. 7.19.



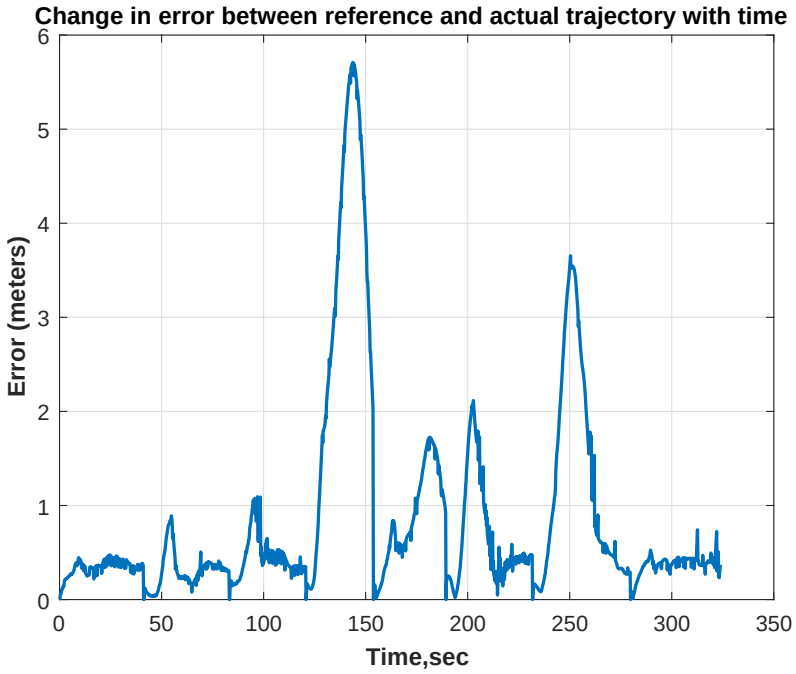
(a) Test case-1



(b) Test case-2

Figure 7.18: Reference and actual path of the rover with only GPS for state-feedback

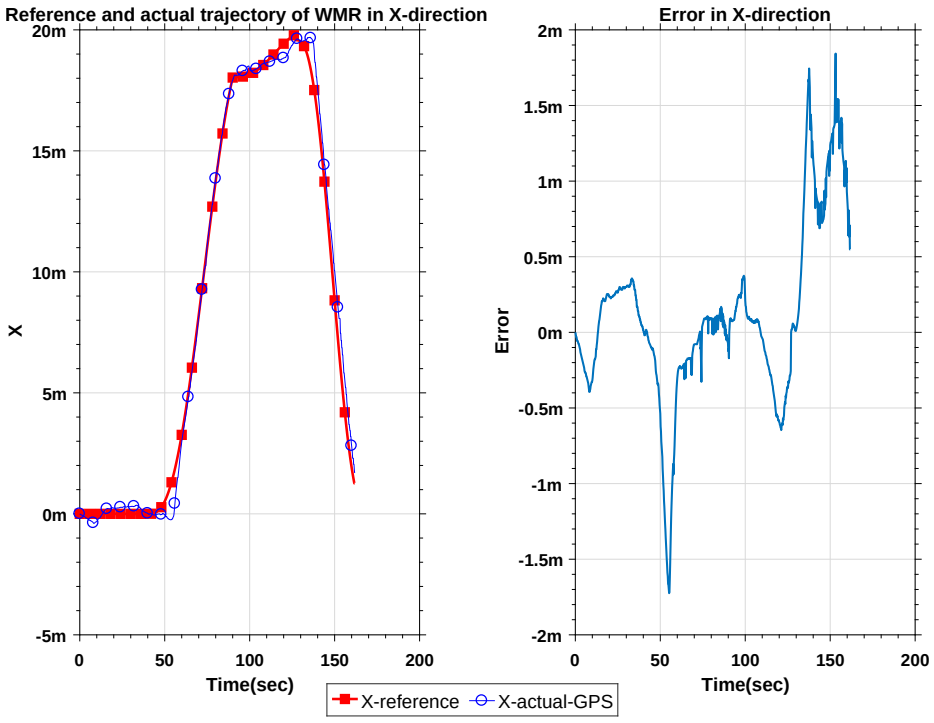
(a) Test case-1



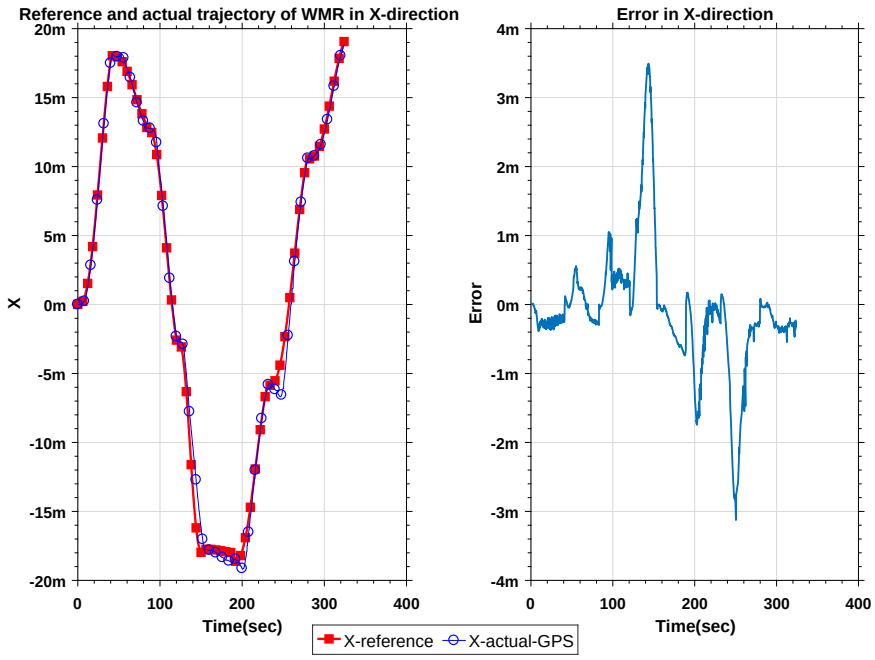
(b) Test case-2

Figure 7.19: Error in position between reference and actual trajectory of the rover with only GPS for state-feedback

As there will be a small jumps in GPS measurements, the heading angle that is calculated from GPS measurements will also have a small jumps. So, as the calculated heading angle jumps, the control signals generated by the controller will also have jumps, to minimize the error. The jumps in heading angle can be seen in Fig. 7.22a and the commanded angular velocity by the controller can be seen in Figs. 7.23a and 7.24a where it is evident that the controller is trying to minimize the heading error.

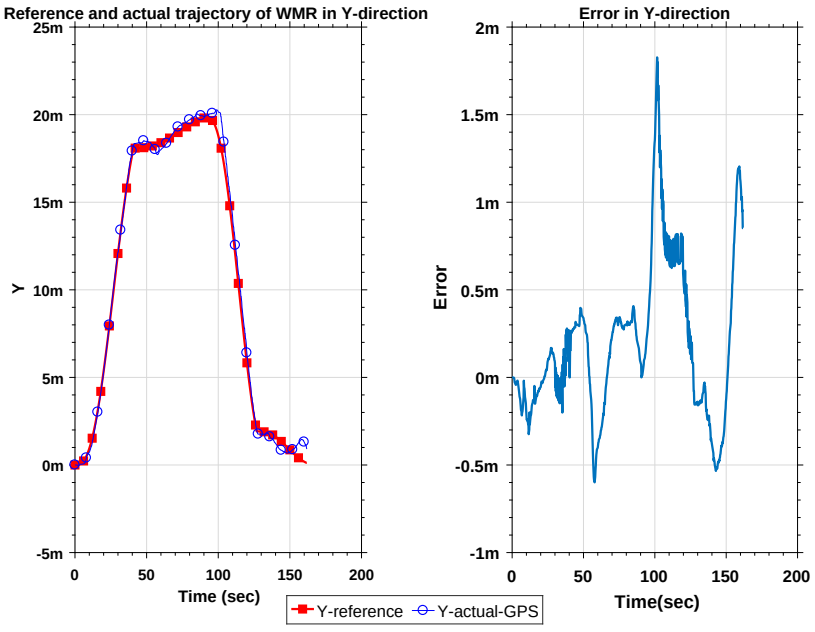


(a) Test case-1

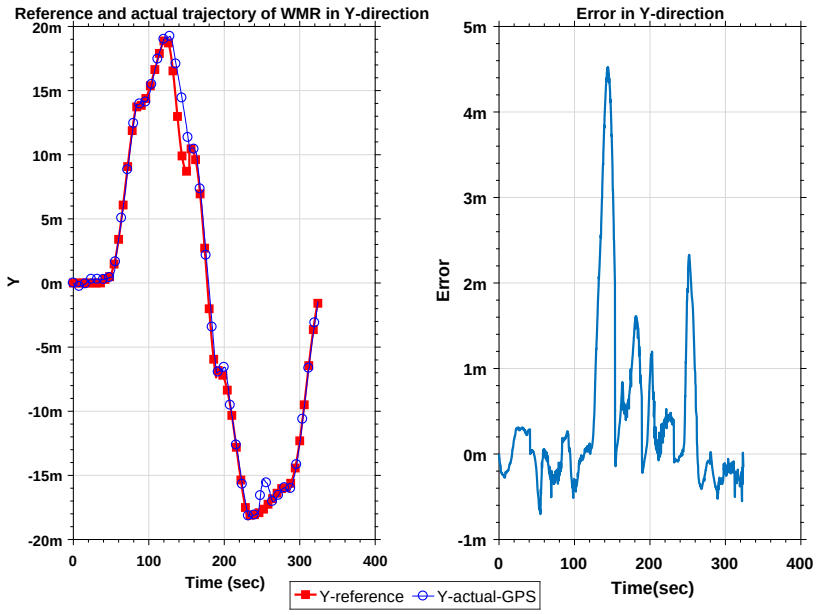


(b) Test case-2

Figure 7.20: Reference and actual trajectory of the rover in X-direction with only GPS for state-feedback

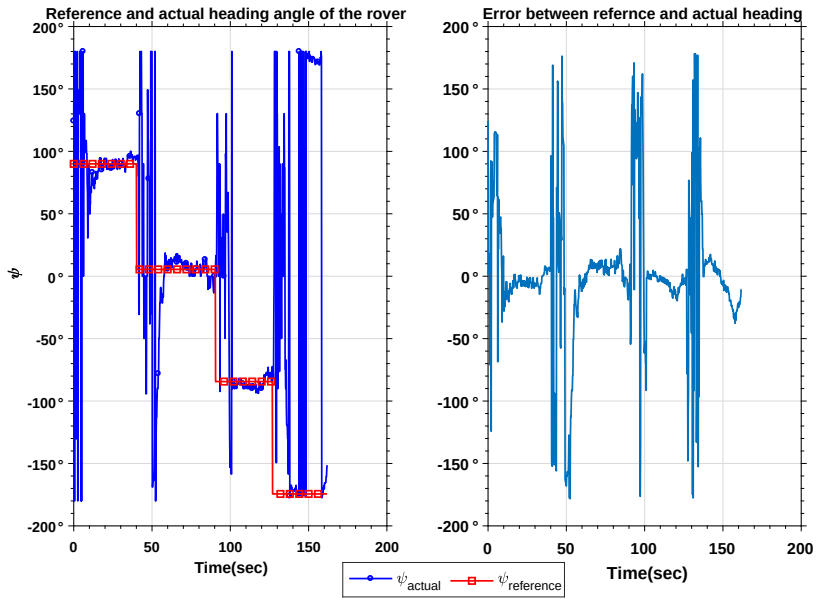


(a) Test case-1

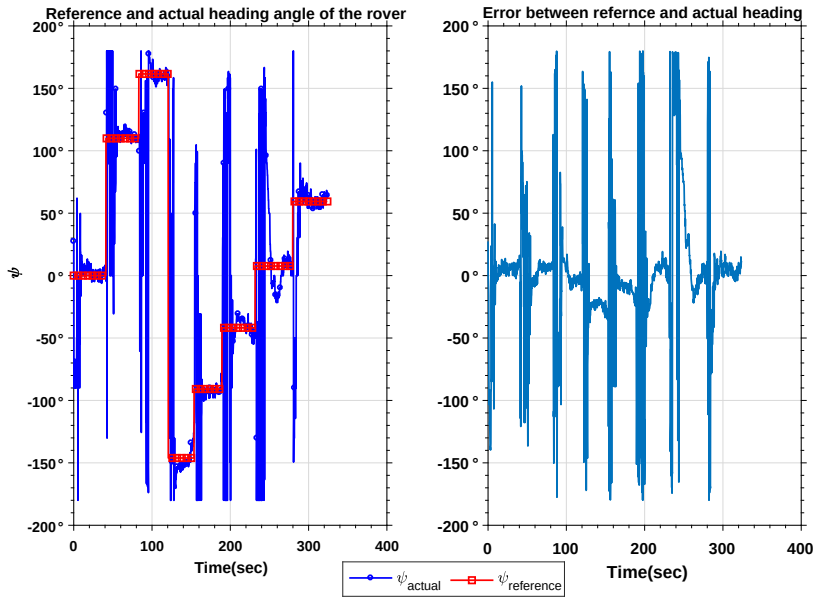


(b) Test case-2

Figure 7.21: Reference and actual trajectory of the rover in Y-direction with only GPS for state-feedback

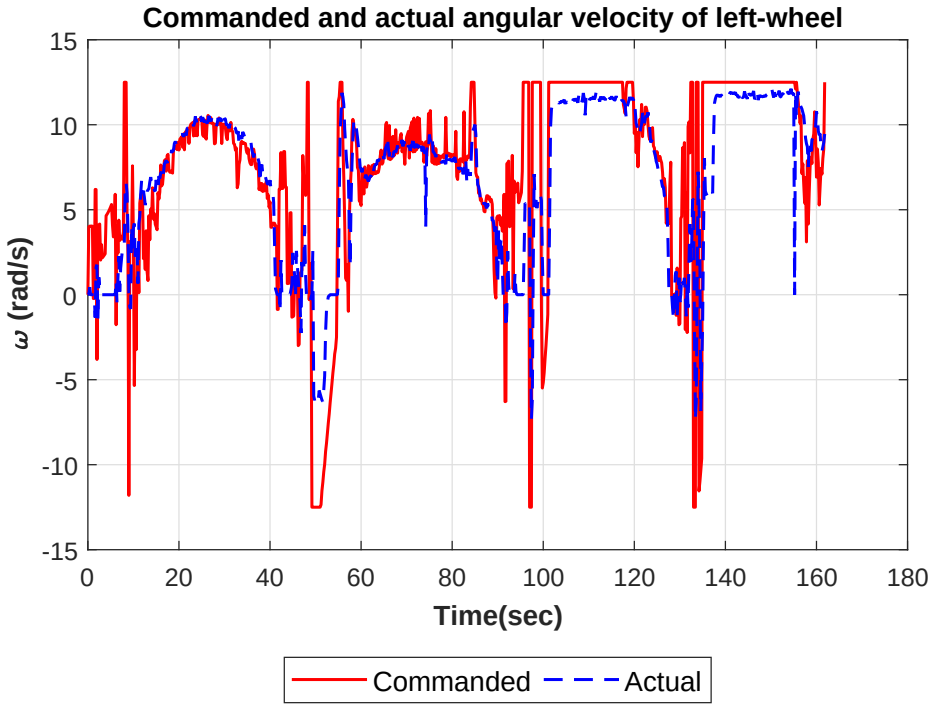


(a) Test case-1

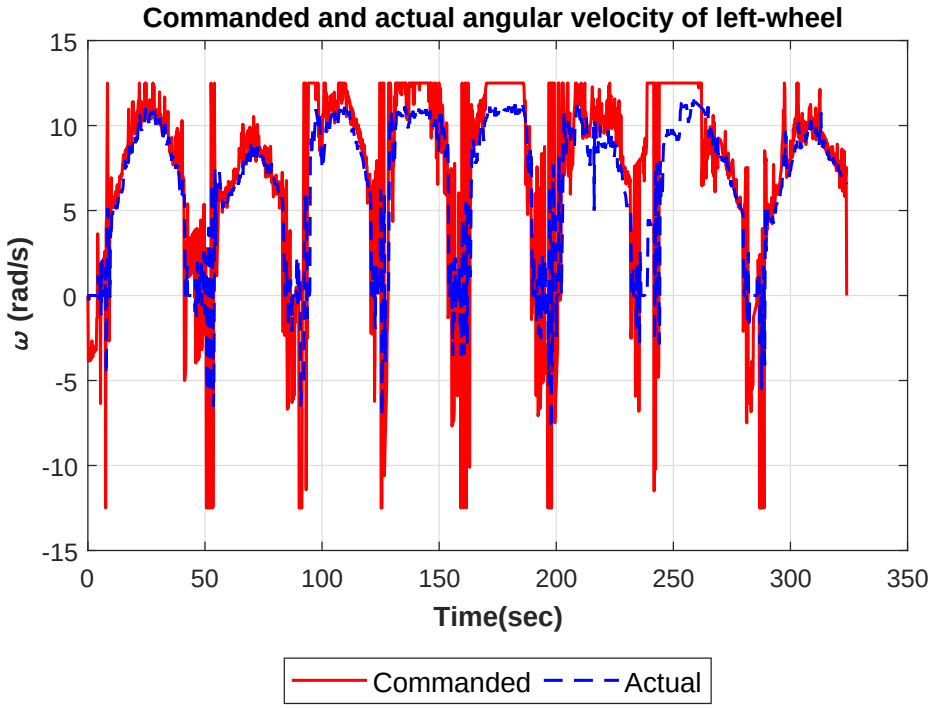


(b) Test case-2

Figure 7.22: Reference and actual heading angle of the rover with only GPS for state-feedback

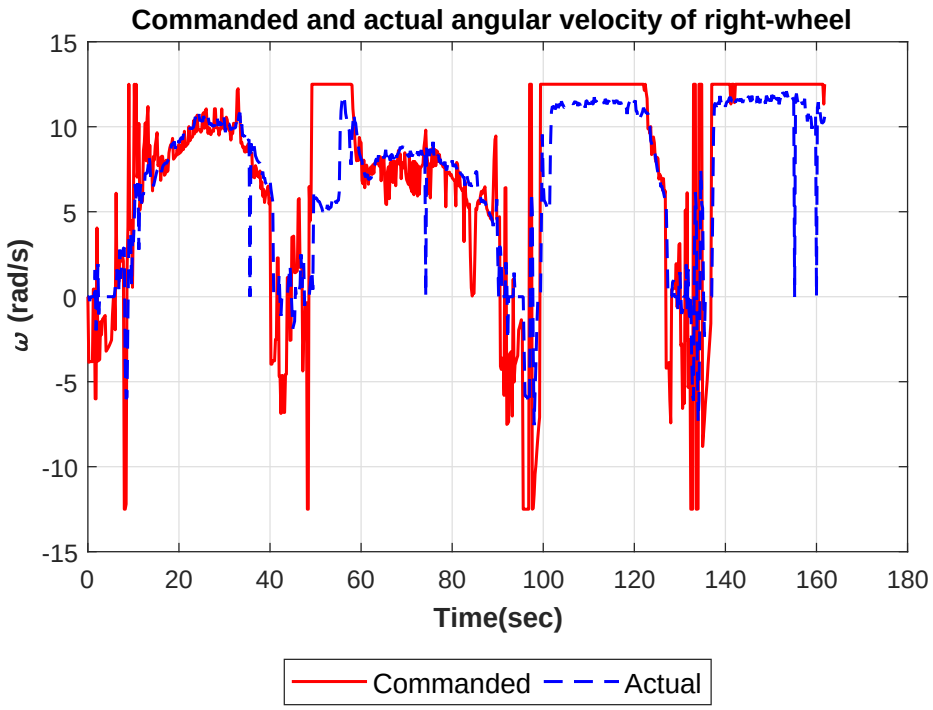


(a) Test case-1

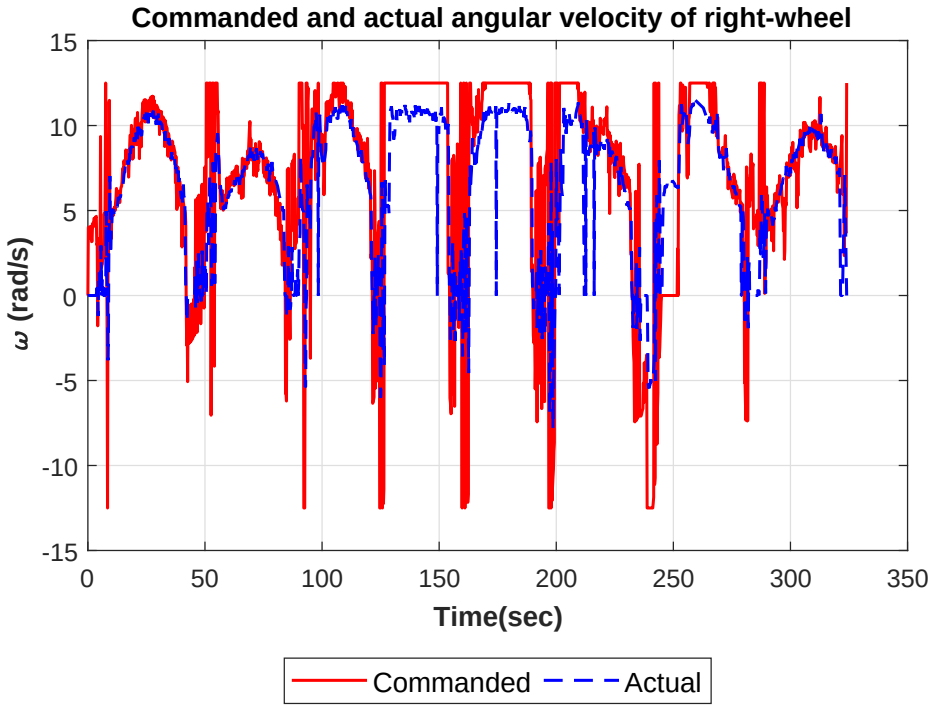


(b) Test case-2

Figure 7.23: Commanded and actual angular velocity (rad/s) of left-wheel with only GPS for state-feedback



(a) Test case-1



(b) Test case-2

Figure 7.24: Commanded and actual angular velocity (rad/s) of right-wheel with only GPS for state-feedback

7.4 Using Circular Trajectory Segments for Turning

For this test case, a constant velocity circular trajectory explained in Sec. 3.4 is used to align the rover towards the next target waypoint when the rover reaches the current waypoint proximity. In-order to create the circular trajectory, a turning radius of 2 m and a constant resultant velocity of 0.5 m/sec is considered. Fig. 7.25 shows the trajectory tracking of the rover and the circular trajectory stitching for the test case-1.

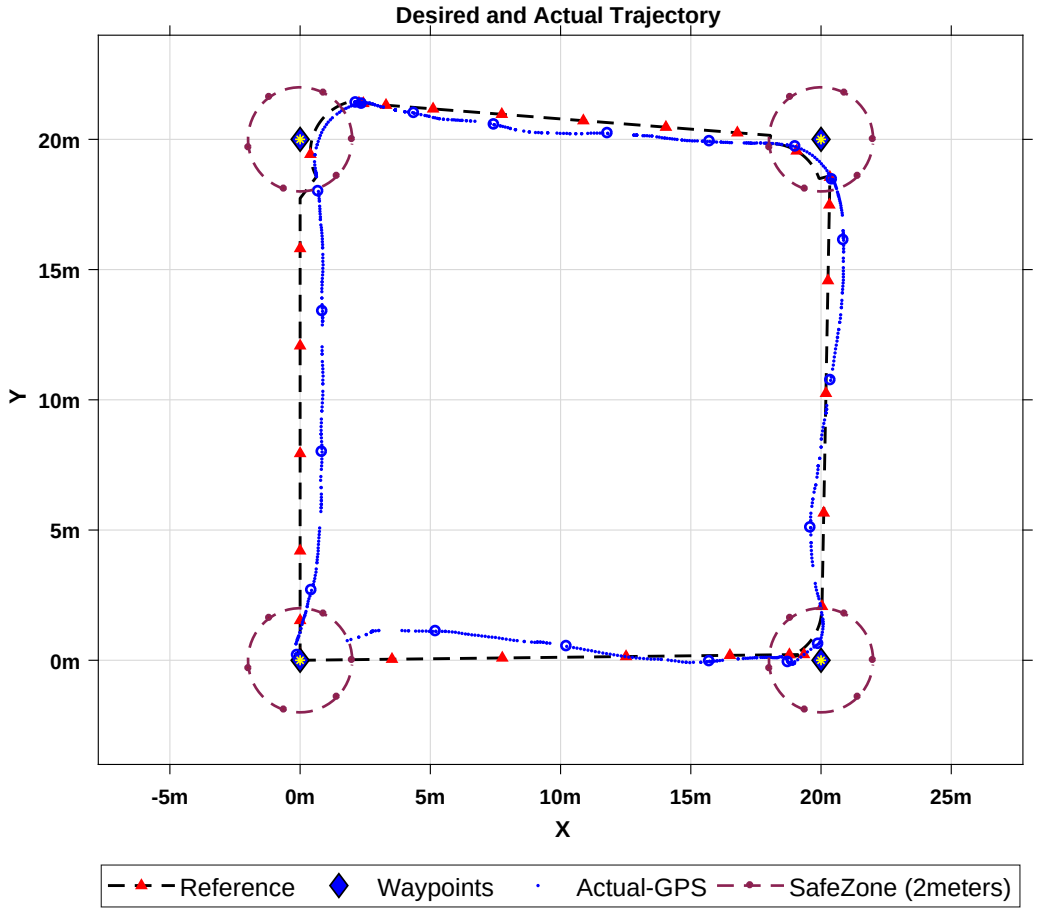


Figure 7.25: Reference and actual path of the rover for the test case-1 with GPS and compass along with circular trajectory stitching

From Fig. 7.25 we can see the stitched circular trajectory at the waypoint transition. As the circular trajectory is created to align the rover towards the next waypoint, the rover followed the circular trajectory and turned gradually rather than turning in-place as this is the case in previous results. The trajectory tracking of the rover in x, y direction is shown in Figs. 7.26, 7.27. From Fig. 7.28 we can see that as the circular trajectory segments are used to align the rover towards the next desired heading, there is a gradual change in the heading angle. Because of this the rover was able to turn smoothly and reach the goal position at the end.

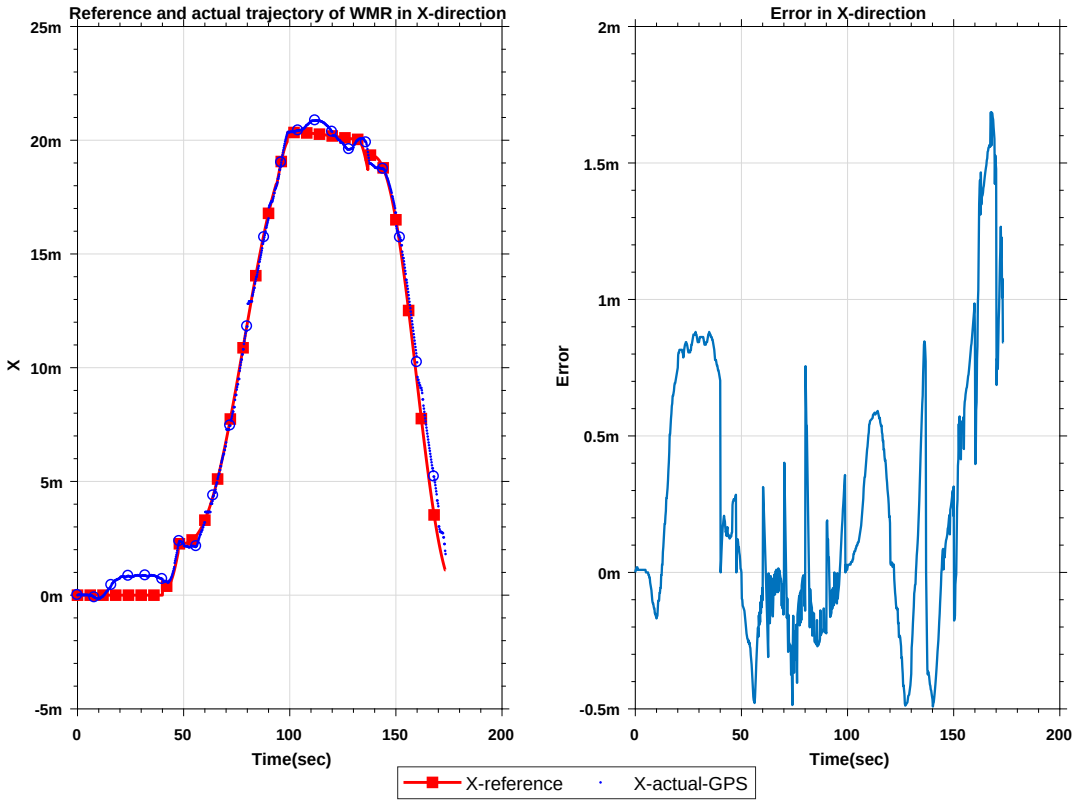


Figure 7.26: Reference and actual trajectory of the rover in X-direction with GPS and compass along with circular trajectory stitching

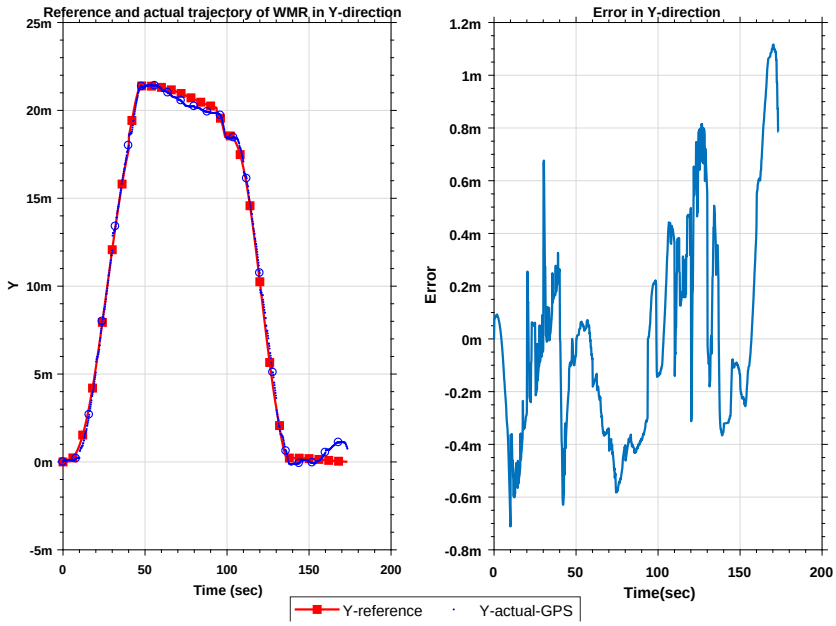


Figure 7.27: Reference and actual trajectory of the rover in Y-direction with GPS and compass along with circular trajectory stitching

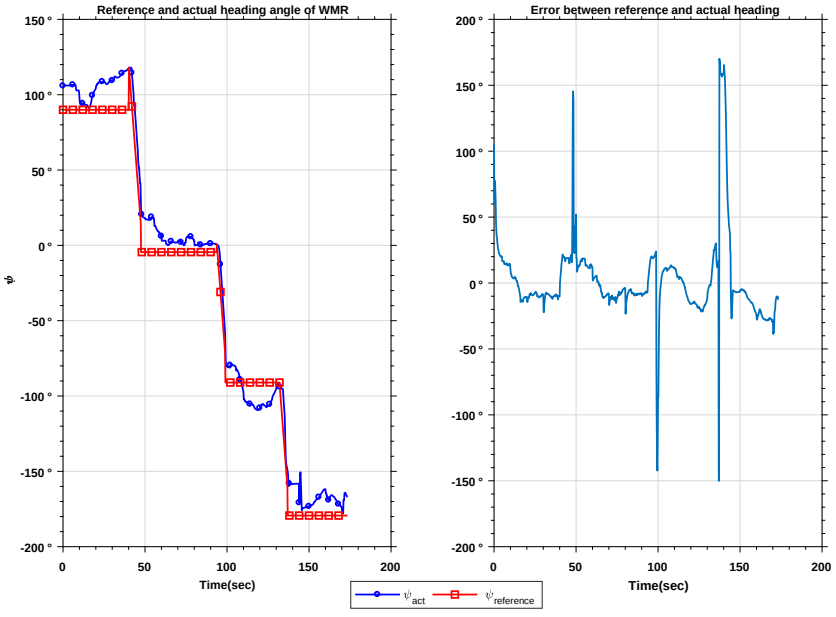


Figure 7.28: Reference and actual heading angle of the rover with GPS and compass along with circular trajectory stitching

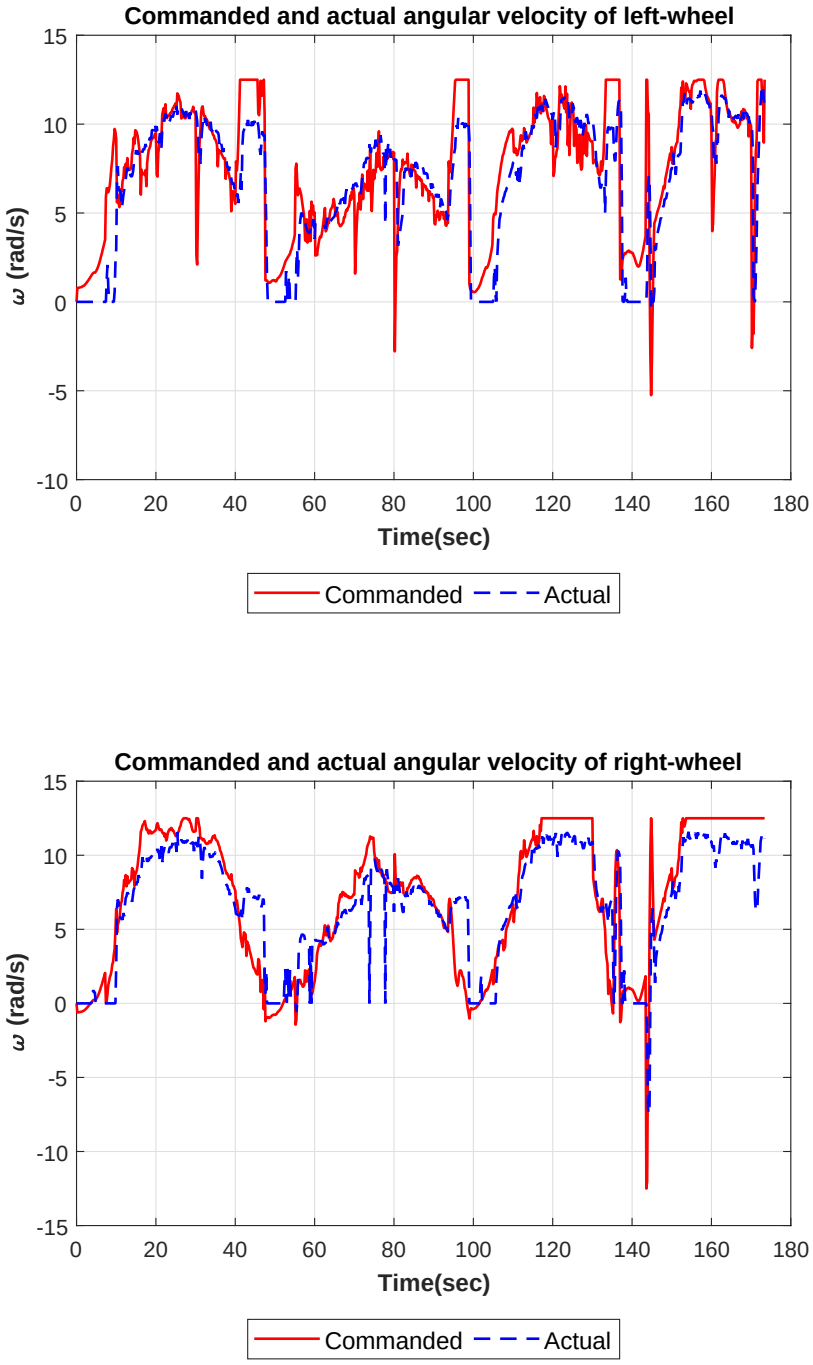


Figure 7.29: Commanded and actual angular velocity (rad/s) of the wheels with GPS and compass along with circular trajectory stitching

CHAPTER 8

Summary, Conclusions and Future Work

8.1 Summary and Conclusions

This thesis presents a novel optimal trajectory generation scheme based on the constraints of the system. The trajectory generation scheme was analyzed experimentally for different acceleration bounds. As the motors that were used on the rover are of high-torque and low speed this sets the physical limits on the rover. So, when an acceleration bounds of more than 0.3 m/sec^2 were used, the final time that was obtained was not enough to track the trajectory as this requires more velocity than the existing motors can provide. This can be rectified by increasing the final time for the trajectory or using more powerful motors. As increasing the final time for the trajectory corresponds to selecting a lower acceleration bound, an acceleration bound below 0.3 m/sec^2 was selected to generate the reference trajectory.

The nonlinear guidance law was verified experimentally. The gains associated with the guidance law were tuned based on the trajectory tracking performance and it was observed that higher gains are needed for reducing the heading angle error than to reduce the positional error. It was observed that when the gains associated with minimizing the position errors are more than 0.5, the commanded wheel speeds are saturated.

The time-delays associated with the cyber-physical system framework which was used to test the trajectory generation and guidance law were thoroughly investigated. It was noticed that the minimum time-delay associated with the network communication in the outdoor environment was about 150 milliseconds. The control scheme

was tested with different time-delays and was found to be robust to network delays of upto 350 milliseconds, without loss in performance. The performance degradation in terms of waypoint proximity was observed to be similar for delays larger than 600 milliseconds.

8.2 Future Work

The work presented in this thesis can be used as a foundation for the future research on unmanned ground vehicles. The communication architecture developed for the rover can be used to test various Guidance, Navigation, and Control laws. Some future studies include,

- Design and verify continuous trajectory generation with system constraints
- Vehicle-to-vehicle communication for co-operative mission planning
- Design of control law to account for time-delays

Appendices

Appendix A

Lyapunov Stability Analysis

For a function to be considered a valid Lyapunov function it must be positive definite, its derivative must be negative definite and both the function and its derivative must be equal to zero at the equilibrium points (i.e. the origin of the system) [35].

With that in mind, a candidate Lyapunov function is chosen as

$$V = \frac{1}{2}\left(e_x^2 + e_y^2 + e_\psi^2\right), \tag{A.1}$$

which is positive definite and is only equal to zero at the origin. The time derivative of V then is

$$\dot{V} = e_x\dot{e}_x + e_y\dot{e}_y + e_\psi\dot{e}_\psi. \tag{A.2}$$

The stability of the system will be shown in two parts. First, stability will be shown without the singularity avoidance algorithm. Then, stability of the system with the singularity avoidance values will be given.

Proof. No singularity avoidance: When $||\hat{v}|| > \epsilon$, for some $\epsilon \ll 1$, and using the wheel commands given in Eq. 4.36 and 4.36, it follows that the heading angle error term in equation A.2 simplifies to

$$\begin{aligned} e_\psi\dot{e}_\psi &= e_\psi\left(\dot{\psi} - \dot{\psi}_{des}\right) = e_\psi\left[\frac{r}{b}\left(\omega_R - \omega_L\right) - \dot{\psi}_{des}\right] \\ &= e_\psi\left[\frac{r}{b}\left(\frac{b}{r}\left(\dot{\psi}_{des} - \lambda_\psi e_\psi\right)\right) - \dot{\psi}_{des}\right] \\ &= -\lambda_\psi e_\psi^2. \end{aligned}$$

This dictates that $\psi \to \psi_{des}$ as $t \to \infty$. Therefore, as $\psi \to \psi_{des}$, the position error terms in Eq. A.2 simplify to

$$\begin{aligned} e_x \dot{e}_x &= e_x \left(\frac{r}{2} (\omega_L + \omega_R) \cos(\psi_{des}) - \dot{x}_r \right) = e_x (\hat{v} \cos(\psi_{des}) - \dot{x}_r) \\ &= e_x ((\dot{x}_r - \lambda_x e_x) - \dot{x}_r) \\ &= -\lambda_x e_x^2, \end{aligned}$$

and

$$\begin{aligned} e_y \dot{e}_y &= e_y \left(\frac{r}{2} (\omega_L + \omega_R) \sin(\psi_{des}) - \dot{y}_r \right) = e_y (\hat{v} \sin(\psi_{des}) - \dot{y}_r) \\ &= e_y ((\dot{y}_r - \lambda_y e_y) - \dot{y}_r) \\ &= -\lambda_y e_y^2. \end{aligned}$$

Hence, the result in Eq. A.2 becomes

$$\dot{V} = -\lambda_x e_x^2 - \lambda_y e_y^2 - \lambda_\psi e_\psi^2, \tag{A.3}$$

which is negative definite and only equal to zero at the origin. Therefore, Eq. A.1 is a valid Lyapunov function. Furthermore, the system given in Eq. 2.14 with the wheel commands in Eq. 4.36 and 4.36 is stable and the errors will converge to zero as $t \to \infty$ [35]. □

Proof. With singularity avoidance: When the value of $||\hat{v}|| \leq \epsilon$, for some positive $\epsilon \ll 1$, the singularity avoidance algorithm is implemented. From the candidate Lyapunov function derivative and the wheel commands, the heading angle error terms simplify to

$$\begin{aligned} e_\psi \dot{e}_\psi &= e_\psi \left(\dot{\psi} - \dot{\psi}_{des} \right) = e_\psi \left[\frac{r}{b} (\omega_R - \omega_L) - 0 \right] \\ &= e_\psi \left[\frac{r}{b} \left(\frac{b}{r} (0 - \lambda_\psi e_\psi) \right) - 0 \right] \\ &= -\lambda_\psi e_\psi^2, \end{aligned}$$

since $\dot{\psi}_{des} = 0$ by definition in the algorithm. This dictates that $\psi \to \psi_{des}$ where ψ_{des} is the reference value ψ_{ref} by definition. Furthermore as $\psi \to \psi_{des}$, the position tracking errors reduce to

$$\begin{aligned} e_x \dot{e}_x &= e_x \left(\frac{r}{2} (\omega_L + \omega_R) \cos(\psi_{des}) - \dot{x}_{ref} \right) = e_x (\hat{v} \cos(\psi_{des}) - \dot{x}_{ref}) \\ &= e_x ((\dot{x}_{ref} - \lambda_x e_x) - \dot{x}_{ref}) \\ &= -\lambda_x e_x^2, \end{aligned}$$

and

$$\begin{aligned} e_y \dot{e}_y &= e_y \left(\frac{r}{2} (\omega_L + \omega_R) \sin(\psi_{des}) - \dot{y}_{ref} \right) = e_y (\hat{v} \sin(\psi_{des}) - \dot{y}_{ref}) \\ &= e_y ((\dot{y}_{ref} - \lambda_y e_y) - \dot{y}_{ref}) \\ &= -\lambda_y e_y^2. \end{aligned}$$

Thus, the result in Eq. A.2 becomes

$$\dot{V} = -\lambda_x e_x^2 - \lambda_y e_y^2 - \lambda_\psi e_\psi^2, \tag{A.4}$$

which is negative definite and equal to zero at the origin. Therefore, the system is stable and the errors will converge to zero as $t \to \infty$. □

Appendix B

Software Configuration For Data Transfer Between Components On The Rover

Before going further into the setup and configuration, first we need to discuss about the framework that is used for the setup, which is called Robot Operating System (ROS). ROS is a flexible framework for writing robot software. It is a framework with collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. The following mentioned concepts are needed before going forward and in-depth understanding can be found in [36].

- ROS File-System:
  - Packages: These are the basic unit of the ROS software. it contains all the runtime processes called nodes, libraries, package configuration etc.
  - Messages: Messages are the type of information that is sent from one ros process to other ros process. These are usually located inside ros package with the file extension ".msg". These files are located inside the package as,

    `ros_package_name/msg/<message-name>.msg`
  - Services: Services are kind of a request/reply interaction between processes. The service information is usually stored with the file extension ".srv". These files are located inside the package as,

    `ros_package_name/srv/<service-name>.srv`

- ROS Computation-Graph Level: This the framework that ROS uses for handling all the computation and talking between other processes. The main concepts in the computation graph are,
  - Nodes: These are the processes that performs the computation.
  - Master: The ROS Master provides the registration and lookup to the rest of the nodes. Nodes will not be able to find each other, exchange messages, or invoke services without a ROS Master.
  - Parameter Server: The parameter server allows you to keep the data to be stored in a central location. This is part of the ROS Master.
  - Topics: Each message in ROS is transported using named buses called topics. When a node sends a message through a topic, then we can say the node is publishing a topic. When a node receives a message through a topic, then we can say that the node is subscribing to a topic.
  - Bags: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, which can be difficult to collect but is necessary for developing and testing robot algorithms.

B.1 Configuration for data transfer between Pixhawk and Odroid

Once the Mavros software is installed from the ROS respository(MAVROS) and after connecting the on-board computer to the pixhawk using an FTDI cable, the next step is to configure the Mavros package as follows,

- Get the port number that the pixhawk is connected to, let's say that the port number is `/dev/ttyUSB0`.

- Connect the pixhawk to the Mission planner and change the baudrate of the TELEM2 port to 921600(this can be done by setting `SERIAL1_baud` under configuration -> Full Parameter List)
- Now open the file "apm2.launch" by going to the launch folder of the Mavros package, and change the `fcu_url` parameter to `/dev/ttyUSB0:921600`
- Now launch the apm2.launch to connect to pixhawk and access the data.

B.2 Setup for data transfer between Arduino and Odroid

Once the Arduino IDE is installed on the odroid, the ROS package rosserial needs to be installed. The installation procedure is given on clearpath-robotics website(rosserial).

After that running the following command will connect the odroid to arduino using ROS framework.

```
1    rosrun rosserial_python serial_node.py /dev/ttyACM0
```

where `/dev/ttyACM0` is the port to which arduino is connected to.

B.3 Reading quadrature encoders using Arduino

Before going on to the problem of how to read the quadrature encoders off the motors, one needs to know how the quadrature encoders works. Quadrature encoders are a kind of rotary encoders where the direction of rotation of the motor can be determined. Quadrature encoders consists of two channels, channel A and channel B. The output from both the channels will either be HIGH(5 V) or LOW(0 V). Usually channel B has a phase lag of 90^o compared to A. So when the signal from channel A leads channel B, it says that the motor is rotating in FORWARD(clockwise) direction. Similarly when the signal from channel B leads channel A, it represents that the motor

is rotating in REVERSE(anti-clockwise) direction. And the speed of the motor can be obtained by looking at the rate of the change of these signals from HIGH to LOW. Fig. B.1 shows the signals state of channel A and B when the motor is turning in both the directions.

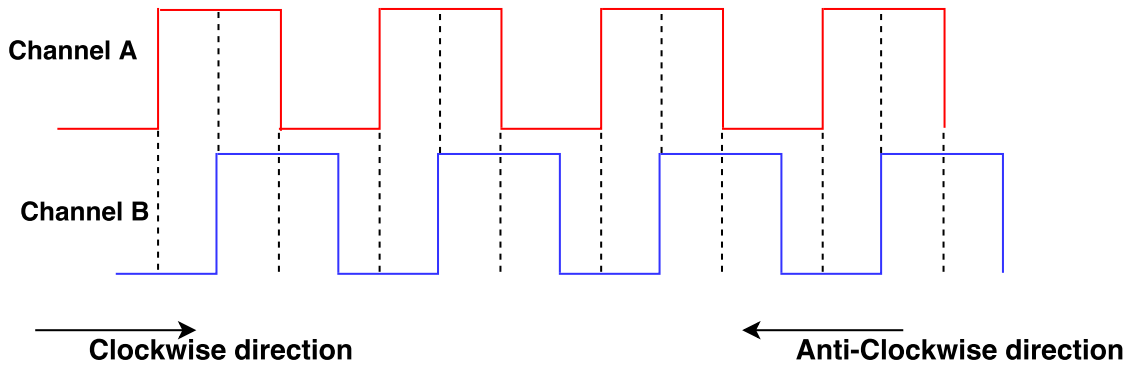


Figure B.1: Signal state of the quadrature encoder based on the direction of the motor movement

Now to increment/decrement the encoder count based on the speed of the motor, we need a way to increment/decrement count whenever a signal change occurs in either of the channels. To account for this the "Hardware Interrupts" functionality available inside arduino mega is used. The main advantage of using hardware interrupts is that, it eliminates the need for manual signal polling to check whether the signal state is changed. So whenever a signal change occurs in either of the channels the hardware interrupt calls a predefined function and based on the previous signal state of the channels, the encoder count is either incremented or decremented. The algorithm that was used to increment/decrement the encoder count is given in Table. B.1.

Previous signal state		Current signal state		Count
A	B	A	B	
0	0	1	0	+1
0	0	0	1	- 1
1	0	1	1	+1
1	0	0	0	- 1
1	1	0	1	+1
1	1	1	0	- 1
0	1	0	0	+1
0	1	1	1	- 1

Table B.1: Algorithm to increment/decrement encoder ticks based on the motor direction, where 1 represents signal is HIGH and 0 represents signal is LOW

In this case whenever the signal change occurs the count of the encoder in incremented/decremented based on the motor turning direction and the state of the channels A,B based on the Table B.1

B.4 Network Communication Setup

ROS gives us the ability to communicate between different nodes over-the-network with a very minimalistic configuration. let's say that we want two system(two robots or a robot and a computer) needed to be connected so that the data from the computer can be sent to robot and the data on the rover can be accessed by computer. Let the tow systems be system1 and system2 . For generality let's say that the system1 network IP address is system1-IP and the system2 network IP address is system2-IP. Before proceeding further we need to select one of the system as a Master so that the ROS Master will be running on that machine. Let the Master system in this case be system1, so other systems will be connected to system1.

Before launching the "roscore" two variables needed to be set on all the machines. They are,

- **ROS_MASTER_URI**: IP address where the ROS Master should run or is running and the syntax for setting this variable is,

```
export ROS_MASTER_URI=http://<MASTER-IP>:11311
```

- **ROS_IP**: IP address of the current machine. The syntax for setting this variable is,

```
export ROS_IP=<system-IP>
```

Example: For connecting system1 and system2 with the ROS-Master running on system1,

On System1

```
export ROS_IP=<system1-IP>
export ROS_MASTER_URI=http://<system1-IP>:11311
```

On System2

```
export ROS_IP=<system2-IP>
export ROS_MASTER_URI=http://<system1-IP>:11311
```

Once the above procedure is followed, the communication between system1 and system2 will be established and both the systems will be able to access the data on each other machines.

B.5 Network Communication setup between Rover and MATLAB

As the MATLAB is being used for simulating control signals and sending it to the rover, the MATLAB needed to be connected to the ROS-Master on the rover. The command "rosinit" in MATLAB needed to be run to start the ROS on MATLAB. To connect MATLAB to existing ROS-Master the IP address of the machine where ROS-Master is running need to be specified to "rosinit".

Example: To connect to ROS-Master at IP address <ROS-Master-IP>,

```
rosinit('ROS-Master-IP')
```

Appendix C

Obtaining state-feedback from different sensors

C.1 Converting global GPS co-ordinates to local cartesian co-ordinates of rover

Since GPS gives the position information of the rover in the global co-ordinate frame, these measurements need to be transformed to local navigation frame(ENU) in-order to localize the rover. To transform these GPS measurements to ENU frame, the GPS measurements are first transformed to Earth-Centered-Earth-Fixed(ECEF) frame by using WGS-84 ellipsoidal model given by Eq's. C.1-C.4. Then by taking the first ECEF co-ordinate as a starting position, these ECEF co-ordinates are tranformed to ENU frame using Eq's. C.5-C.6.

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \tag{C.1}$$

$$X_{ECEF} = (N + h) \cos \phi \cos \lambda \tag{C.2}$$

$$Y_{ECEF} = (N + h) \cos \phi \sin \lambda \tag{C.3}$$

$$Z_{ECEF} = (N(1 - e^2) + h) \sin \phi \tag{C.4}$$

$$\mathbf{P_{NED}} = \begin{pmatrix} -\sin\phi\cos\lambda & -\sin\phi\sin\lambda & \cos\phi \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\phi\cos\lambda & -\cos\phi\sin\lambda & -\sin\phi \end{pmatrix} \mathbf{\Delta ECEF} \tag{C.5}$$

$$\mathbf{P_{ENU}} = R_{NED}^{ENU}\, \mathbf{P_{NED}} \tag{C.6}$$

where N is the length of the normal to the ellipsoid, X_{ECEF} is the X co-ordinate of the rover in ECEF frame, Y_{ECEF} is the Y co-ordinate of the rover in ECEF frame,

Z_{ECEF} is the Z co-ordinate of the rover in ECEF frame, ϕ, λ, h are the latitude, longitude and altitude at the current rover position, a is the semi-major axis of the earth$(6,378,137.0\ m)$, b is the semi-minor axis of earth$(6,356,752.3142\ m)$, e is the eccentricity of the earth, R_{NED}^{ENU} is the rotation matrix from NED to ENU frame, $P_{NED} = [X_{NED}, Y_{NED}, Z_{NED}]^T$, $P_{ENU} = [X_{ENU}, Y_{ENU}, Z_{ENU}]^T$ and $\Delta ECEF = [ECEF_{current} - ECEF_{ini}]$.

C.2 Relationship between wheel angular velocity to motor PWM

In-order to command the motors, an appropriate relationship between the angular velocity and motor PWM voltage signals is required. To get this relationship, a series of PWM values are selected. Then these selected PWM values are applied to motors with the rover on the ground for a time span of δt and the traveled distance by rover is measured. By using PWM and the distance traveled, the angular velocity of the wheel(ω) is calculated using equation C.7 and the obtained relationship between the angular velocity of the wheels to motor PWM signal is illustrated in figure C.1.

$$\omega = \frac{\Delta s}{r\delta t}. \tag{C.7}$$

where Δs is the distance travelled.

Fig. C.1 shows the relationship between the angular velocity of the wheels to motor PWM signal.

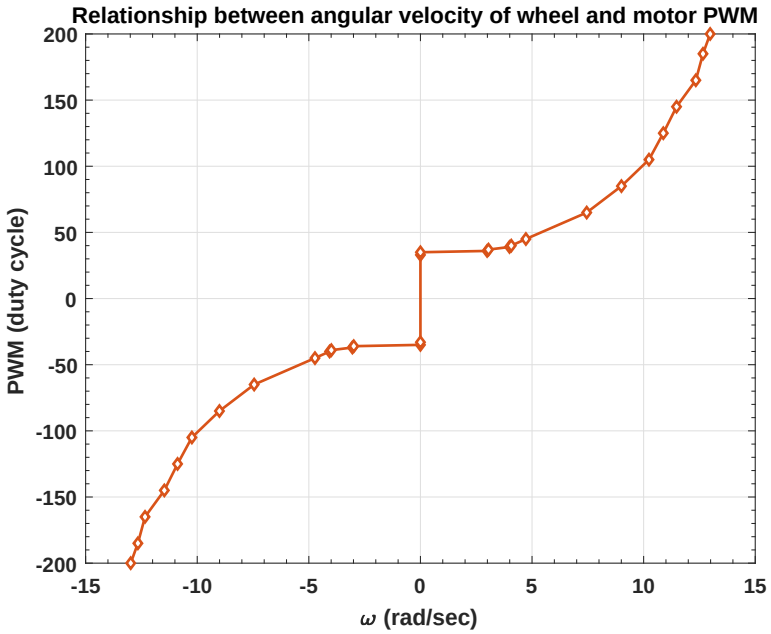


Figure C.1: Obtained relationship between angular velocity of wheel and motor PWM

C.3 Converting motor encoder count to local cartesian position of rover

In-order to determine the position of the rover using the wheel encoders, first the actual angular velocity of the wheels need to be determined. Position from the quadrature encoders is obtained by counting the number of encoder pulses and integrating the state differential equations defined in Eq. C.9 using the initial condition of the rover$(x_0,\ y_0,\ \psi_0)$. It should be noted that the encoders have a tendency to drift linearly with time because of wheel slippage or micro-controller hangups or wheels are not perfectly rigid. Because of this reason encoders were not too reliable, especially when the rover has to travel long distances. However, for short distances, a position accuracy of 1 cm can be achieved. This technique is popularly known as dead reckoning.

The actual angular velocity of the vehicle is calculated by using the encoders and Eq. C.8,

$$\omega_{actual} = \frac{(\delta_{ET})}{r\gamma_{ET}\delta t}, \tag{C.8}$$

where r is the radius of the wheel (m), δ_{ET} is the number of ticks that is elapsed since the last known encoder measurement, γ_{ET} is the number of encoder ticks that would obtained if the rover travels for a distance of 1 m, and δt is the time elapsed since the last known encoder measurement.

$$\begin{aligned} \dot{X}_{act} &= \frac{r}{2}(\omega_{r\{act\}} \ + \ \omega_{l\{act\}}) \cos\psi_{act} \\ \dot{Y}_{act} &= \frac{r}{2}(\omega_{r\{act\}} \ + \ \omega_{r\{act\}}) \sin\psi_{act} \end{aligned} \tag{C.9}$$

where, $\omega_{r\{act\}}$, $\omega_{l\{act\}}$ are the right and left actual wheel angular velocities respectively and ψ_{act} is the actual heading angle of the rover.

REFERENCES

[1] Craig, J. J., *Introduction to Robotics: Mechanics and Control*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd ed., 1989.

[2] Dubins, L. E., "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, Vol. 79, No. 3, 1957, pp. 497–516.

[3] Reeds, J. and Shepp, L., "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, Vol. 145, No. 2, 1990, pp. 367–393.

[4] Wilburn, J. N., Perhinschi, M. G., and Wilburn, B. K., "Implementation of composite clothoid paths for continuous curvature trajectory generation for UAVs," *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013, p. 5230.

[5] Tsai, P.-S., Wang, L.-S., Chang, F.-R., and Wu, T.-F., "Systematic backstepping design for b-spline trajectory tracking control of the mobile robot in hierarchical model," *Networking, Sensing and Control, 2004 IEEE International Conference on*, Vol. 2, IEEE, 2004, pp. 713–718.

[6] Jeon, J. W. and Ha, Y. Y., "A generalized approach for the acceleration and deceleration of industrial robots and CNC machine tools," *IEEE transactions on industrial electronics*, Vol. 47, No. 1, 2000, pp. 133–139.

[7] Khalsa, D. and Mahoney, B., "High performance motion control trajectory commands based on the convolution integral and digital filtering," *Proceedings of International Conference on Intelligent Motion*, 1990, pp. 54–61.

[8] Dixon, W., Dawson, D., Zergeroglum, E., and Zhang, F., "Robust tracking and regulation control for mobile robots," *Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on*, Vol. 2, IEEE, 1999, pp. 1015–1020.

[9] Huang, J., Wen, C., Wang, W., and Jiang, Z.-P., "Adaptive stabilization and tracking control of a nonholonomic mobile robot with input saturation and disturbance," *Systems & Control Letters*, Vol. 62, No. 3, 2013, pp. 234–241.

[10] Wilson, D. G. and Robinett, I., "Robust adaptive backstepping control for a nonholonomic mobile robot," *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, Vol. 5, IEEE, 2001, pp. 3241–3245.

[11] Fierro, R. and Lewis, F. L., "Control of a nonholonomic mobile robot: backstepping kinematics into dynamics," *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, Vol. 4, IEEE, 1995, pp. 3805–3810.

[12] Quillen, P., Subbarao, K., and Muñoz, J., "Guidance and Control of a Mobile Robot via Numerical Navigation Functions and Backstepping for Planetary Exploration Missions," *AIAA SPACE 2016*, 2016, p. 5237.

[13] Ostafew, C. J., Schoellig, A. P., and Barfoot, T. D., "Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments," *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 4029–4036.

[14] Martins, F. N., Celeste, W. C., Carelli, R., Sarcinelli-Filho, M., and Bastos-Filho, T. F., "An adaptive dynamic controller for autonomous mobile robot trajectory tracking," *Control Engineering Practice*, Vol. 16, No. 11, 2008, pp. 1354–1363.

[15] Buccieri, D., Perritaz, D., Mullhaupt, P., Jiang, Z.-P., and Bonvin, D., "Velocity-scheduling control for a unicycle mobile robot: Theory and experiments," *IEEE Transactions on Robotics*, Vol. 25, No. 2, 2009, pp. 451–458.

[16] De Luca, A., Oriolo, G., and Samson, C., "Feedback control of a nonholonomic car-like robot," *Robot motion planning and control*, 1998, pp. 171–253.

[17] Isidori, A., *Nonlinear control systems*, Springer Science & Business Media, 2013.

[18] Chen, C., He, Y., Bu, C., Han, J., and Zhang, X., "Quartic bezier curve based trajectory generation for autonomous vehicles with curvature and velocity constraints," *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 6108–6113.

[19] Kimia, B. B., Frankel, I., and Popescu, A.-M., "Euler spiral for shape completion," *International journal of computer vision*, Vol. 54, No. 1, 2003, pp. 159–182.

[20] Pourboghrat, F., "Exponential stabilization of nonholonomic mobile robots," *Computers & Electrical Engineering*, Vol. 28, No. 5, 2002, pp. 349–359.

[21] Kanayama, Y., Nilipour, A., and Lelm, C. A., "A locomotion control method for autonomous vehicles," *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, IEEE, 1988, pp. 1315–1317.

[22] LaValle, S. M., *Planning algorithms*, Cambridge university press, 2006.

[23] Lepetič, M., Klančar, G., Škrjanc, I., Matko, D., and Potočnik, B., "Time optimal path planning considering acceleration limits," *Robotics and Autonomous Systems*, Vol. 45, No. 3, 2003, pp. 199–210.

[24] Kanayama, Y., Kimura, Y., Miyazaki, F., and Noguchi, T., "A stable tracking control method for an autonomous mobile robot," *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, IEEE, 1990, pp. 384–389.

[25] Samson, C., "Time-varying feedback stabilization of car-like wheeled mobile robots," *The International journal of robotics research*, Vol. 12, No. 1, 1993, pp. 55–64.

[26] Klančar, G. and Škrjanc, I., "Tracking-error model-based predictive control for mobile robots in real time," *Robotics and autonomous systems*, Vol. 55, No. 6, 2007, pp. 460–469.

[27] Kokotovic, P. V., "The joy of feedback: nonlinear and adaptive," *IEEE Control systems*, Vol. 12, No. 3, 1992, pp. 7–17.

[28] Fossen, T. I. and Strand, J. P., "Tutorial on nonlinear backstepping: applications to ship control," *Modeling, identification and control*, Vol. 20, No. 2, 1999, pp. 83.

[29] Godbole, A., VNV, M., Quillen, P., and Subbarao, K., "Optimal Trajectory Design and Control of a Planetary Exploration Rover," *AIAA SPACE 2017*, 2017.

[30] Pixhawk, "Pixhawk specifications," Available at `https://pixhawk.org/modules/pixhawk`.

[31] "Robot Operating System," Available at `http://www.ros.org`.

[32] "MAVROS – MAVLink extendable communication node for ROS with proxy for Ground Control Station." Available at `http://wiki.ros.org/mavros`.

[33] "ROSSerial protocol for wrapping ROS serialized messages over a serial port or network socket." Available at `http://wiki.ros.org/rosserial`.

[34] MATLAB, "Robotics System ToolBox," Available at `https://www.mathworks.com/products/robotics.html`.

[35] Slotine, J.-J. E., Li, W., et al., *Applied nonlinear control*, Vol. 199, Prentice hall Englewood Cliffs, NJ, 1991.

[36] Joseph, L., *Mastering ROS for robotics programming*, Packt Publishing Ltd, 2015.

BIOGRAPHICAL STATEMENT

V. N. V. Murali obtained his bachelor's degree in Mechanical Engineering from GITAM University, India. After graduation, he joined The University of Texas at Arlington to pursue an M.S. program in Mechanical Engineering. His areas of interest include Guidance, Navigation, and Control of unmanned vehicles and control of dynamical systems.