Simulation and Control of Nonholonomic Differential Drive Platforms

A THESIS

SUBMITTED TO THE FACULTY OF THE

UNIVERSITY OF MINNESOTA

BY

Scott Robert Norr

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Advisor: Jiann-Shiou Yang, PhD

December 2017

Acknowledgements

It is with great relief that I write this final paragraph, signifying the end of a long journey that has taken far too long. So many barriers cross the path of those who attempt this goal. Without the help, guidance and support of those around us, far fewer would attain the prize.

First and foremost, my most humble gratitude to my advisor, Jiann-Shiou Yang, a person of vast intellect and even larger patience. Second, deep and loving thanks to my darling wife, Lisa, who always, always, always has my back. Third, to my committee members, Mohammed Hasan and Howard Mooers, sound men willing to turn a calm, blind eye to folly; also to Stan Burns who hired me on at UMD and has always supported my efforts. And last, to Shey Peterson, the heart and soul of the EE department, without whom we haven’t a hope of getting anything done.

Abstract

This thesis explores the feasibility of applying non-linear control techniques to a small, inexpensive robot with limited computing ability. First, a fundamental basis for the kinematic description of Differential Drive Mobile Robot (DDMR)s is presented. The dynamics of wheeled robots, actuated by permanent magnet DC motors are developed. The state space of DDMR platforms is explored and found to be non-linear. A non-linear control law, based on a conference paper by Kanayama, is developed and determined to be bounded by a Lyapunov function and asymptotically stable. Using MATLAB, the entire closed-loop DDMR system is modeled with time-stepped difference equations. Methods for tuning the control gains are explored. A prototype robot is constructed from inexpensive parts, including a very modest 8-bit processor. A significant limitation in the robot is discovered, that being the inexpensive wheel encoders. Even with the limitation, reasonable correlation between the physical robot and the simulated robot is observed**.** The limitation does not hinder the robot's ability to successfully implement a non-linear control scheme. The MATLAB simulation and physical robot show good agreement. The control law is shown to be practical for small, inexpensive robotic platforms.

Table of Contents

List of Tables

List of Figures

List of Variables

Model Variables:

AH (amp-hours) – The ampere-hour capacity of the battery

F (N m s) – The frictional coefficient matrix of the entire robotic platform

GRi (unitless) – The gear ratio of the i^{th} motor

J (N m^2) – The inertial moment matrix of the entire robotic platform

Kai and Kti – The armature and torque constants of the i^{th} motor. In SI units, these are the same value

L (meters) – A measurement from the robot center of rotation to the center of the ith wheel

Lai (henry) – The armature reactance of the i^{th} DC motor

m (kg)– Mass of the complete robotic platform

P – the center of rotation in the differential drive robot

R() – The standard rotation matrix in the X-Y plane

Rai (ohms) – The armature resistance of the i^{th} DC motor

Rb (ohms) – The internal resistance of the battery

ri (meters) – The radius of the i^{th} wheel

Vb (volts) – The battery voltage for the mobile robotic platform

Vdc (volts) – The voltage applied to the drive motors

v_g (m s^{-1}) – The linear velocity of the goal (tracked) object

v_R (m s^{-1}) – The linear velocity of the robot

ω_g (rad s^{-1}) – The angular velocity of the goal (tracked) object

ω_R (rad s^{-1}) – The angular velocity of the robot, rotating around point, P

State Variables:

Iai (amperes) – The armature current of the ith DC motor

φi (radian s^{-1}) – The angular frequency of the ith motor armature (also used for ith wheel angular velocity)

Ei (meters, radians) – The [x, y, ϴ] pose of the robotic platform in the global (inertial) frame

E_R (meters, radians) – The [x, y, ϴ] pose of the robotic platform in the robot frame

ϴ (radians) – The angular displacement between robot and global frames

Input Variables:

Eg – The position vector of the tracked (goal) object in the global frame

v_g, ω_g – The linear and angular velocities of the tracked object

Vdci (volts) – The applied voltage of the ith motor

Output Variable:

Ei – The position vector of the robotic platform in the global frame

Acronyms:

DDMR – Differential Drive Mobile Robot

DOF – Degree of Freedom

DDOF – Differential Degree of Freedom

Chapter 1

Introduction

The world is poised on the edge of a revolution. Rapid advances in machine learning coupled with continuing improvements in materials and manufacturing techniques foretell the coming age of autonomy. At the forefront, artificially intelligent automobiles are logging large numbers of on the road miles and appear ready for market. Indeed, a recent article in Forbes Magazine predicts a rapid deployment in self-driving automobiles, with millions on the road in as little as three years [1].

A typical self-driving car is manufactured with precision tolerances using materials designed for high reliability. It has thousands of dollars invested in both long-range and short-range sensors, feeding highway information to multiple microprocessors running sophisticated control algorithms. The machine learning algorithms that oversee all hardware and keep passengers safe, have taken decades to develop and test. They run on high capacity computing platforms at high data rates.

But what of the opposite end of the mobile autonomy scale? That is the small, cheaply manufactured wheeled robotic platforms, usually with two drive wheels and limited computing capability. In this category there are currently high-end markets for products such as robotic vacuum cleaners, such as the Roomba™. These markets are likely to expand to include other household duties and chores outside the home, such as sidewalk cleaning, light landscaping and garden tending. As an example, a new product called the Tertill (turtle), can identify and destroy small weeds in a garden plot [2]. Devices such as these are now being 3-D printed and may become common and even indispensable.

The differential drive wheeled mobile robot (DDMR) is ubiquitous in this kind of market. Nearly every budding roboticist has developed such a platform, since they are quite easily constructed. Surprisingly then, considering its popularity, the platform is nonholonomic and its control behavior is non-linear. A great many control schemes have been developed to address the problem, using several approaches. Among them, linearization around an operating point, state feedback linearization and non-linear algorithms bounded to stable regions by Lyapunov functions [3].

This thesis explores the application of a sophisticated tracking control algorithm to a simple DDMR, built with inexpensive parts and controlled by a small 8-bit microprocessor. Floating point operations are required for most of the computational overhead. Therefore, the iterative loop that implements the control algorithm is necessarily slow. The following questions are explored and answered: Can sophisticated control algorithms be implemented on a small, slow 8-bit processor? Are modern non-linear control techniques robust enough to work well on small, inexpensive robots, where precision parts are not used? Is the lack of detailed engineering information and poor precision of inexpensive parts a barrier to their use or can a few simple measurements provide enough information to create useful models?

To answer these questions kinematic and dynamic models of a small robotic platform are developed, a non-linear control scheme to implement tracking control is customized and applied and a custom simulator is coded in MATLAB for rapid prototyping of control parameters and prediction of robot behavior. In addition, a simple set of measurements are presented to determine the pertinent physical parameters of a small inexpensive robot. The parameters are coded into the simulator and its results are tested against the physical robot performance. Sensitivities are run on the simulator to determine suitable gain constants for the controller. The same gain constants are programmed into the robot and its performance is compared to the predictions of the simulator.

Chapter 2

Mobile Robotic Platform Kinematics

Consider the differential drive wheeled mobile robot (DDMR) as pictured in Figure 2-1. It has two drive motors, attached to two main wheels through a gear reduction. A third point of ground contact is made at the rear of the vehicle using a ball castor or low-friction skid. For the purposes of modeling and analysis, the robot can be considered confined to the X-Y Plane.

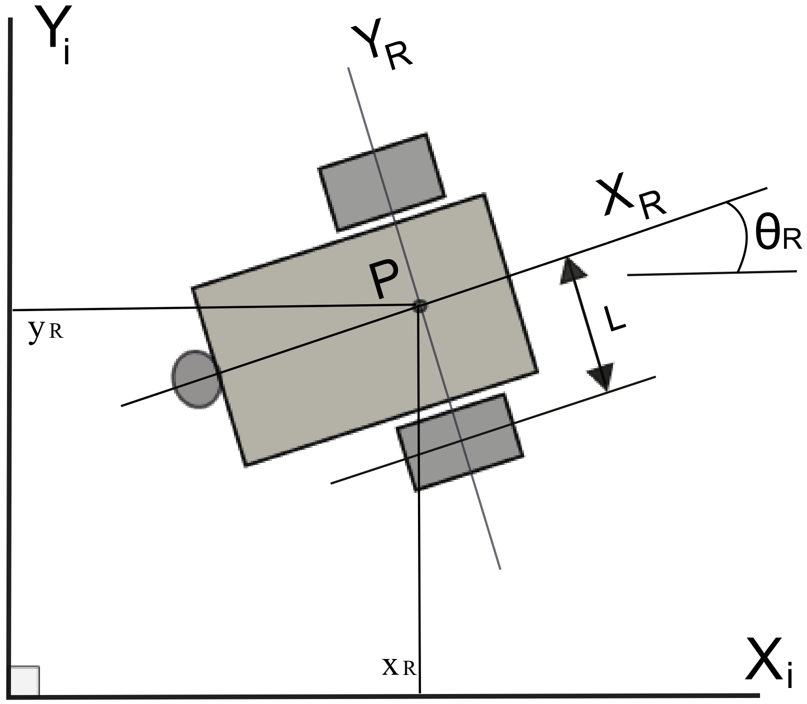


FIGURE 2-1: Differential Drive Wheeled Mobile Robot (DDMR)

As indicated in the figure, a point, **P**, on the robot is designated. It is chosen to be at the midpoint between the two wheels, on a line drawn through the wheel axles, such that any difference in angular velocity between the two drive wheels will cause a rotation about **P**. Extending from **P**, the basis **[X_R, Y_R]** forms a set of axes that define the robot's local frame of reference. Any arbitrary basis **[X_i, Y_i]** forms a set of axes in an inertial frame, serving as the global frame of reference. The angular difference between the two frames is designated as **Θ_R**. It is therefore possible to define the point **P** in the global frame as a position vector, using the notation from Siegwart [4]**.**

Position Vector in Global Frame:

$$\mathbf{E_i = [\ x_R\ \ y_R\ \ \Theta_R\ ]^T} \qquad \mathbf{(1)}$$

Thus, in the global frame, the robot has 3 degrees of freedom (**DOF**), position in both **X** and **Y** and an orientation angle, **ϴ**. In the robot frame, however, the DDMR is constrained by its fixed, non-steerable wheels from moving laterally (in the **Y_R** direction). When motive force is delivered to the drive wheels, a forward velocity produces movement in the **X_R** direction only. That velocity being the arithmetic average of the velocity contribution from each wheel. Any difference in wheel contribution results in a net rotational velocity about the point, **P**. Figure 2-2 illustrates the concept.

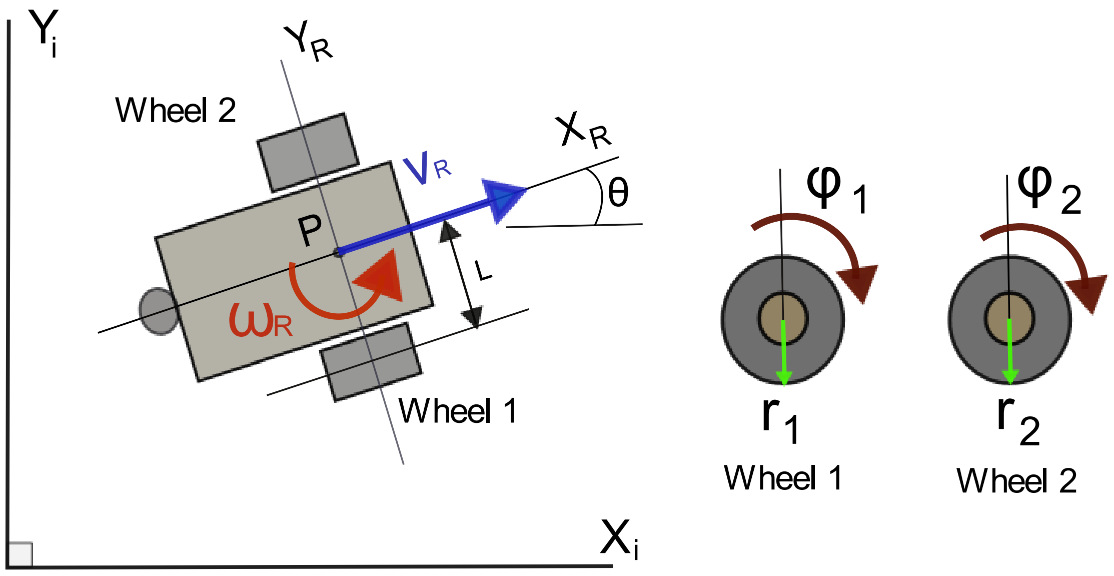


FIGURE 2-2: Linear and Angular Velocities in the DDMR

When examining the robot in motion in its own frame, it becomes clear that there are only two differential degrees of freedom (**DDOF**): $\mathbf{v_R}$, velocity as a rate of change in $\mathbf{X_R}$ and $\boldsymbol{\omega_R}$, angular velocity as a rate of change in $\boldsymbol{\Theta}$. This is summarized in the following equations:

Velocity Vector, Robot Frame: $$\dot{E}_R = [\ \dot{x}_R\ \ \dot{y}_R\ \ \dot{\Theta}_R\ ]^T = [\ v_R\ \ 0\ \ \omega_R\ ]^T \qquad (2)$$

Velocity Vector in Global Frame: $$\dot{E}_i = R(\Theta_R)\,\dot{E}_R = [\ v_x\ \ v_y\ \ \omega_R\ ]^T \qquad (3)$$

Rotation Matrix: $$R(\ ) = \begin{bmatrix} cos(\ ) & -sin(\ ) & 0 \\ sin(\ ) & cos(\ ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4)$$

The rotation matrix, **R(ϴ),** in Equation (3) is the standard 3x3 matrix for rotations of angle **ϴ** in the X-Y plane, around the Z-axis. It projects velocities in the robot frame onto the bases of the inertial frame. Since the DDOF = 2 which is less than the DOF of 3, the robot is classified as nonholonomic [4]. In the DDMR, the implication is that the fixed position of the two drive wheels place a nonholonomic kinematic constraint on movement in the Y_R direction**.**

The relationship between wheel motion and the quantities found in the velocity vector are algebraic and simple. The angular velocity of each wheel, $\boldsymbol{\varphi_i}$, translates into a linear velocity proportional to the wheel radius, $\mathbf{r_i}$**.** The robot linear velocity is the arithmetic average of the velocity contributions from each wheel:

$$\textbf{Robot Linear Velocity:} \quad \mathbf{v_R = (r_1 \varphi_1 + r_2 \varphi_2) / 2} \qquad \textbf{(5a)}$$

Likewise, the difference in angular velocity of wheel 1 with respect to wheel 2 creates a rate of rotation around the point **P**, defined as $\boldsymbol{\omega_R} = \dot{\theta}_{\mathbf{R}}$, dependent on the radius, **L**, the displacement radius from the wheel to the center, **P**.

$$\textbf{Robot Angular Velocity:} \quad \boldsymbol{\omega_R} = \mathbf{(r_1 \varphi_1 - r_2 \varphi_2) / 2L} \qquad \textbf{(5b)}$$

This then is a sufficient description of the kinematics of the DDMR workspace and the non-linear relationships translating wheel angular velocities into linear velocities in the inertial plane. Equations (5a) and (5b) can be written as a single equation (6), showing the kinematic matrix, **A_k**, relating wheel velocities to robot velocities [4]:

$$\dot{E}_R = \begin{bmatrix} v_R \\ 0 \\ \omega_R \end{bmatrix} = A_k\varphi = \begin{bmatrix} \frac{r_1}{2} & \frac{r_2}{2} \\ 0 & 0 \\ \frac{r1}{2L} & \frac{-r2}{2L} \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \end{bmatrix} \qquad \textbf{(6)}$$

Chapter 3

Differential Drive Mobile Robot Dynamics

The kinematics described in Chapter 2 solidify the relationship between wheel velocities, $\boldsymbol{\varphi}$, and position in the inertial frame, $\mathbf{E_i}$. Further work is required to mathematically relate some form of control input to the wheel velocity outputs, $\boldsymbol{\varphi}$. For the DDMR under study, wheel actuation is achieved using gear-reduced permanent magnet DC motors (PMDC), due to their low cost and linear performance. PMDCs possess a constant speed characteristic that is proportional to applied voltage. The most practical control input therefore is a pair of voltage signals, Vdc, one for each drive motor. Such input voltages are easily and efficiently varied using a motor driver circuit employing pulse-width modulation (PWM) under the control of the 8-bit microcontroller.

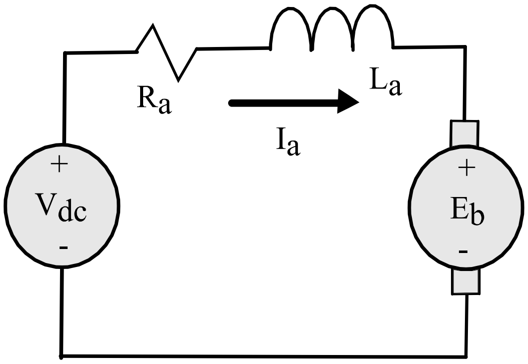
To model the open loop state space of the DDMR it is necessary to develop the mechanics of this electro-mechanical system. From classical mechanics, the sum of the torques equals the product of the moment of inertia and angular acceleration, $\boldsymbol{\Sigma\ \tau = J\ \alpha}$. Expanding that concept to represent all the mechanical torques acting on the robot, Equation (7) looks similar to those found in many journal papers, including Fiero [5].

$$\mathbf{M}\dot{\boldsymbol{\omega}} + \mathbf{C}\boldsymbol{\omega} + \mathbf{F}\boldsymbol{\omega} + \mathbf{G} = \mathbf{B}\boldsymbol{\tau} - \mathbf{A}\boldsymbol{\lambda} \qquad \mathbf{(7)}$$

In (7), **M** is the matrix of inertial moments, **C** is the matric of Coriolis forces, **F** is the matrix of frictional forces, **G** is the gravitational vector, **B** describes the transformation/application of input torques and **A** describes the kinematic constraints of the platform. However, it is important to remember the single nonholonomic constraint in the DDMR, preventing movement in $\mathbf{Y_R}$. If acceleration is limited for robot movement, such that the drive wheels are never allowed to slip, the Coriolis forces are negligible and matrices **C** and **A** can be [null]. In addition, modeling the robot in the X-Y plane, normal to the gravitational vector, the gravitational forces are represented within the frictional force matrix, **F**. Lastly, a change of variable is necessary. Omega, **ω,** is already used to representation rotation around P in the robot frame. Instead, the letter phi, **φ**, is used to represent the angular velocity of the wheels. Equation (7) can then be simplified to the following expression with only minor loss of accuracy for the DDMR, as discussed in Chapter 5 of Kuo [6].

$$\mathbf{M}\dot{\boldsymbol{\varphi}} + \mathbf{F}\,\boldsymbol{\varphi} = \mathbf{B}\boldsymbol{\tau} \tag{8}$$

The **τ** in equation (8) represents the mechanical torque input vector developed by the electric motors. Looking at the motors electrically, Kirchhoff's Voltage Law (KVL) can be applied in such a way as to relate the desired control input, $\mathbf{V_{dc}}$, to the electric torque developed. This is described in Figure 3-1, below.



Electric Torque:

$$\boxed{\tau = K_t I_a}$$

KVL:

$$-V_{dc} + I_a R_a + L_a \dot{I}_a + E_b = 0 \quad \text{(ii)}$$

Substitute (i) into (ii):

$$\boxed{L_a \dot{I}_a + R_a I_a = V_{dc} - K_a \varphi}$$

Back EMF:

$$E_b = K_a \varphi \quad \text{(i)}$$

Note: In SI units K_t, the torque constant, equals K_a, the armature constant. φ is the shaft velocity in rad/sec.

FIGURE 3-1: Electrical Dynamics in PMDC Motors

Thus, electric torque is proportional to armature current, which in turn has a differential relationship to input voltage. The applied voltage is also opposed by the back EMF of motor, a by-product of the armature spinning at angular velocity, φ, in the presence of the permanent magnetic field. All that remains is to equate the electrical torque to the mechanical torque under equilibrium conditions and an overall transfer function can be developed, relating input voltages to wheel velocities. This is outlined in the development below, where Equation (8) is combined with the electrical equations of Figure 3-1 to form (9a-9e) and complete the transfer function. Note that the characteristic equation is second order.

$$\text{(Mech. Torque)}\ T_m = J\,\dot{\varphi} + F\,\varphi \quad = \quad T_e = K_t\,I_a\ \text{(Elec. Torque)}$$

$$Tm = Te\text{:}\ \ J\,\dot{\varphi} + F\,\varphi = K_t\,I_a \quad \textbf{(9a)} \qquad L_a\,\dot{I}_a + R_a\,I_a = V_{dc} - K_a\,\varphi \quad \textbf{(9b)}$$

In the Laplace Domain:

$$(J\,\mathbf{s} + F)\,\Phi(\mathbf{s}) = K_t I_a(\mathbf{s}) \quad \textbf{(9c)} \qquad (L_a\mathbf{s} + R_a)\,I_a(\mathbf{s}) = V_{dc} - K_a\Phi(\mathbf{s}) \quad \textbf{(9d)}$$

Combining (9c) with (9d) to form the Open Loop Transfer Function:

$$\frac{\mathbf{\Phi(s)}}{\mathbf{V_{dc}}} = \frac{\mathbf{K_t}}{\mathbf{(\,Js + F\,)(\,L_a s + R_a\,) + K_t K_a}} \qquad \textbf{(9e)}$$

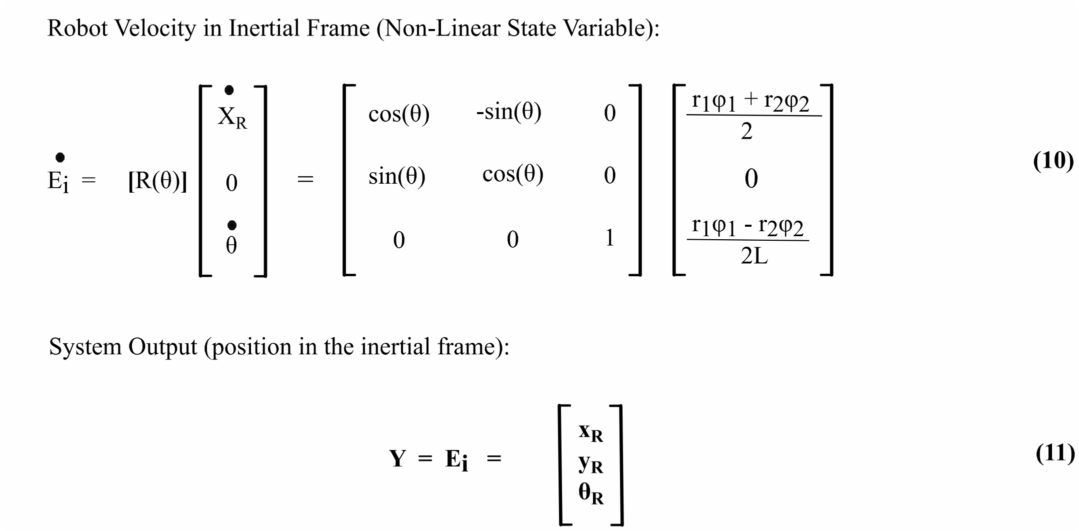
It is now possible to build the state space model in the form, $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$. Each motor has two state variables, an armature current, $\mathbf{I_a}$, and a shaft/wheel angular velocity, $\boldsymbol{\varphi}$, as well as an input variable, $\mathbf{V_{dc}}$. For two drive motors then, a total of four state variables. In addition, two other states, the two DDOF, $\mathbf{v_R}$ and $\boldsymbol{\omega_R}$, are required to translate wheel velocities to robot motion in the inertial frame. Recall that $\mathbf{v} = \dot{\boldsymbol{x}}_\mathbf{R}$ and $\boldsymbol{\omega}_\mathbf{R} = \dot{\theta}_\mathbf{R}$. Also note that, using SI units, $\mathbf{K_a} = \mathbf{K_t}$. These six state equations are linear and are written below in matrix form as Equation (9f).

Even though both motors are attached to the same physical platform and performing work on the same mass, each motor is decoupled from the other, working against it's own moment of inertia, $\mathbf{J_i}$. Consider the case where only one motor is spinning, forcing the robot to rotate in a circle with the opposite wheel as the center point. That is the true inertial moment seen by the motor. Now imagine the superposition of both motors spinning simultaneously, each trying to turn the robot in an opposing direction. The net result is forward motion.

$$\begin{bmatrix} \dot{I}_{a1} \\ \dot{\varphi}_1 \\ \dot{I}_{a2} \\ \dot{\varphi}_2 \\ \dot{X}_R \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{-R_{a1}}{L_{a1}} & \frac{-K_{a1}}{L_{a1}} & 0 & 0 & 0 & 0 \\ \frac{K_{t1}}{J_1} & \frac{-F_1}{J_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{-R_{a2}}{L_{a2}} & \frac{-K_{a2}}{L_{a2}} & 0 & 0 \\ 0 & 0 & \frac{K_{t2}}{J_2} & \frac{-F_2}{J_2} & 0 & 0 \\ 0 & \frac{r_1}{2} & 0 & \frac{r_2}{2} & 0 & 0 \\ 0 & \frac{r_1}{2L} & 0 & \frac{-r_2}{2L} & 0 & 0 \end{bmatrix} \begin{bmatrix} I_{a1} \\ \varphi_1 \\ I_{a2} \\ \varphi_2 \\ X_R \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{V_{dc1}}{L_{a1}} \\ 0 \\ \frac{V_{dc2}}{L_{a2}} \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \textbf{(9f)}$$

The Controllability Matrix, $\mathbf{[B\ AB\ A^2B\ \ldots\ A^5B]}$ for the state space above is full rank for the parameters $\mathbf{J}$, $\mathbf{F}$, $\mathbf{K}$, $\mathbf{R_a}$, $\mathbf{L_a}$ and $\mathbf{L}$ used to model the physical robot. Therefore, the system is controllable. It is important to note however, that while the robot dynamics are linear in the robot frame, there is a non-linear mapping to the inertial frame. The velocity vector in the inertial frame, $\mathbf{\dot{E}_i = R(\Theta)\ \dot{E}_R}$, is dependent on the product of two state variables, $\mathbf{\Theta}$ and $\boldsymbol{\varphi}$, as shown in (10). The two DDOF in the robot frame, $\mathbf{v}$ and $\boldsymbol{\omega}$, are translated via the rotation matrix, $\mathbf{R(\Theta)}$, into 3 DOF in the inertial frame.

Thus, the system is MIMO, with the two input variables of V_{dc1} and V_{dc2}. The three outputs of the system, $\mathbf{x_R}$, $\mathbf{y_R}$ and $\mathbf{\Theta_R}$ populate the position vector in the inertial frame, $\mathbf{E_i}$**,** as shown in (11).



Robot Velocity in Inertial Frame (Non-Linear State Variable):

$$\dot{E}_i = [R(\theta)] \begin{bmatrix} \dot{X}_R \\ 0 \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{r_1\varphi_1 + r_2\varphi_2}{2} \\ 0 \\ \frac{r_1\varphi_1 - r_2\varphi_2}{2L} \end{bmatrix} \qquad \mathbf{(10)}$$

System Output (position in the inertial frame):

$$\mathbf{Y} = \mathbf{E_i} = \begin{bmatrix} \mathbf{x_R} \\ \mathbf{y_R} \\ \mathbf{\theta_R} \end{bmatrix} \qquad \mathbf{(11)}$$

Equation (10) is a non-linear state relationship and will require accommodation during the feedback control design in Chapter 4.

Chapter 4

The Tracking Control Problem

The DDMR has two DDOF, the linear and angular velocities in the robot frame, $\mathbf{v_R}$ and $\boldsymbol{\omega_R}$, which map to three DOF in the inertial frame, $\mathbf{E_i = E_R = [\, x_R\ y_R\ \Theta_R\,]^T}$. The relationship is non-linear, as outlined in equation (10). It is desired to use this robotic platform to track a moving object. It is assumed that the robot has the ability to use on-board sensors to assess the dynamics of the tracked object and accurately provide the following information at zero computation cost: $\mathbf{v_g}$, the linear velocity of the object; $\boldsymbol{\omega_g}$, the angular velocity of the object; and also its pose, $\mathbf{[x_g\ y_g\ \Theta_g]^T}$, in the inertial frame.

This is not a small assumption. The ability to sense the distance, velocity and heading of a moving object is hardly trivial. The monetary cost alone damages the premise of an inexpensive platform. But prices drop continuously. The real barrier is the computational load. It would overwhelm the 8-bit AVR. In any event, given that those measured values are available for use, it is possible to construct an error pose for the robot with respect to the tracked object. Figure 4-1 depicts the geometry of the situation, consistent with Kanayama [7].

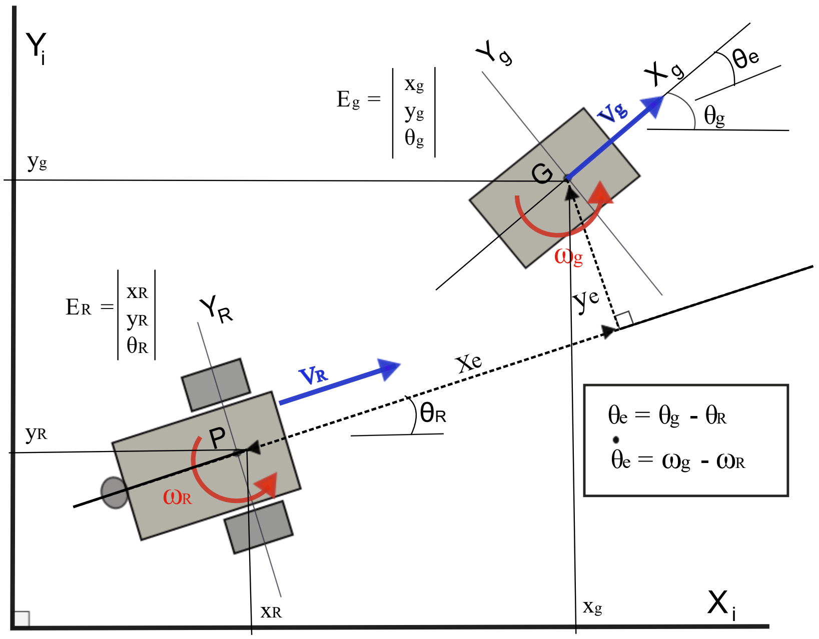


FIGURE 4-1: Tracking Error for the DDMR [7]

In the figure, sensors on the robot, at **P**, are measuring features of the goal object, at **G**, and computing an error pose, **E_e = [x_e y_e Θ_e]^T**. Note that this is calculated in the inertial frame and then projected via rotation onto the robot frame as developed in (12). The derivative of the error pose, **$\dot{E}e$ = [$\dot{x}_e$ $\dot{y}_e$ ω_e]^T**, is also developed as equation (13) [7]**.** This differential error pose is indicative of the manner in which the *velocity* of the error changes, and thus identifies the dependencies necessary for control design.

Tracking Error Pose in Robot Frame:

$$E_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = R^{-1}\theta_R(E_g\text{-}E_R) = \begin{bmatrix} \cos\theta_R & \sin\theta_R & 0 \\ -\sin\theta_R & \cos\theta_R & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_g - x_R \\ y_g - y_R \\ \theta_g - \theta_R \end{bmatrix} \qquad \textbf{(12)}$$

Taking the derivative of **Eₑ** with respect to time, by applying the product and chain rules to (12), yields the following development. Note that the right triangle identity, xsinΘ = ycosΘ is also applied.

$$\dot{E}_e = \begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \frac{d}{dt}\begin{bmatrix} \cos\theta_R(x_g - x_R) + \sin\theta_R(y_g - y_R) \\ -\sin\theta_R(x_g - x_R) + \cos\theta_R(y_g - y_R) \\ \theta_g - \theta_R \end{bmatrix}$$

$$\begin{aligned} \dot{x}_e &= (\dot{x}_g - \dot{x}_R)\cos\theta_R - (x_g - x_R)\dot{\theta}_R\sin\theta_R + (\dot{y}_g - \dot{y}_R)\sin\theta_R + (y_g - y_R)\dot{\theta}_R\cos\theta_R \\ &= y_e\omega_R - v_R + \dot{x}_g\cos\theta_R + \dot{y}_g\sin\theta_R = y_e\omega_R - v_R + \dot{x}_g\cos(\theta_g - \theta_e) + \dot{y}_g\sin(\theta_g - \theta_e) \\ &= y_e\omega_R - v_R + \dot{x}_g(\cos\theta_g\cos\theta_e + \sin\theta_g\sin\theta_e) + \dot{y}_g(\sin\theta_g\cos\theta_e - \cos\theta_g\sin\theta_e) \\ &= y_e\omega_R - v_R + v_g\cos\theta_e \qquad \textbf{(i)} \end{aligned}$$

$$\begin{aligned} \dot{y}_e &= -(\dot{x}_g - \dot{x}_R)\sin\theta_R - (x_g - x_R)\dot{\theta}_R\cos\theta_R + (\dot{y}_g - \dot{y}_R)\cos\theta_R - (y_g - y_R)\dot{\theta}_R\sin\theta_R \\ &= -x_e\omega_R + \dot{x}_R\sin\theta_R - \dot{y}_R\cos\theta_R - \dot{x}_g\sin\theta_R + \dot{y}_g\cos\theta_R \\ &= -x_e\omega_R + \dot{x}_g\sin(\theta_g - \theta_e) - \dot{y}_g\cos(\theta_g - \theta_e) \\ &= -x_e\omega_R + \dot{x}_g(\sin\theta_g\cos\theta_e - \cos\theta_g\sin\theta_e) - \dot{y}_g(\cos\theta_g\cos\theta_e + \sin\theta_g\sin\theta_e) \\ &= -x_e\omega_R + v_g\sin\theta_e \qquad \textbf{(ii)} \end{aligned}$$

$$\dot{\theta}_e = \dot{\theta}_g - \dot{\theta}_R = \omega_g - \omega_R \qquad \textbf{(iii)}$$

$$\dot{E}_e = \begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} y_e\dot{\theta}_R - \dot{x}_R + \dot{x}_g\cos\theta_e \\ -x_e\dot{\theta}_R + \dot{x}_g\sin\theta_e \\ \dot{\theta}_g - \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} y_e\omega_R - v_R + v_g\cos\theta_e \\ -x_e\omega_R + v_g\sin\theta_e \\ \omega_g - \omega_R \end{bmatrix} \qquad \textbf{(13)}$$

As indicated in (13), the error velocity is dependent on the two DDOF, $\mathbf{v_R}$ and $\boldsymbol{\omega_R}$, and five other variables: the linear and angular velocities of the goal object, $\mathbf{v_g}$ and $\boldsymbol{\omega_g}$, and the position error pose, $\mathbf{[\ x_e,\ y_e,\ \Theta_e\ ]^T}$. Incorporating all five of these variables in the control design is desirable and Kanayama uses all five variables in his proposed control law [7]. This control law is an excellent test of the computational capability of an inexpensive, 8-bit microcontroller, requiring floating point calculations and trigonometry. The control law is shown as (14) below.

$$\begin{bmatrix} \mathbf{v_R} \\ \boldsymbol{\omega_R} \end{bmatrix} = \begin{bmatrix} \mathbf{v_g \cos\theta_e + K_x\, x_e} \\ \boldsymbol{\omega_g} \mathbf{\ + v_g\, (K_y\, y_e\ + K_w \sin\theta_e\ )} \end{bmatrix} \qquad \mathbf{(14)}$$

Upon examination, it can be seen that (14) introduces three control gain constants, $\mathbf{K_x}$, $\mathbf{K_y}$ and $\mathbf{K_w}$, with which to tune the control behavior of the system. Kanayama ensures that (14) is bounded and asymptotically stable by finding a Lyapunov Function candidate function. Application of Lyapunov's Theorem is a standard test for stability in nonlinear systems [8]. The candidate function is greater than or equal to zero for any pose error, E_e, and whose derivative is less than or equal to zero for any pose error. The proposed candidate function as per Kanayama is Equation (15).

$$\mathbf{V} = \frac{\mathbf{1}}{\mathbf{2}}\,(\mathbf{x_e}^{\mathbf{2}} + \mathbf{y_e}^{\mathbf{2}}) + \frac{(\mathbf{1} - \mathbf{cos}\boldsymbol{\theta}_\mathbf{e})}{\mathbf{K_y}} \qquad \mathbf{(15)}$$

By the following process, it is shown that V ≥ 0: The pose error, **E_e = [0;0;0]**, can certainly be considered a point of stable equilibrium where V = 0. For any other pose error, V > 0. In addition, it is necessary to ensure that the derivative of V, $\dot{\mathbf{V}}$, is always less than zero or equal to zero at equilibrium. The derivative is calculated below:

By the chain rule:

$$\dot{V} = \dot{x}_e x_e + \dot{y}_e y_e + \frac{\dot{\theta}_e \sin\theta_e}{K_y}$$

Substituting from (13) for $\dot{x}_e$, $\dot{y}_e$, $\dot{\theta}_e$:

$$\dot{V} = (y_e\omega_R - v_R + v_g\cos\theta_e)x_e + (-x_e\omega_R + v_g\sin\theta_e)y_e + \frac{(\omega_g - \omega_R)\sin\theta_e}{K_y}$$

Substituting from (14) for v_R and ω_R :

$$\begin{aligned}\dot{V} = {} & (\omega_g + v_gK_yy_e + v_gK_w\sin\theta_e)x_ey_e - v_gx_e\cos\theta_e + v_gx_e\cos\theta_e - K_xx_e^2 \\ & - (\omega_g + v_gK_yy_e + v_gK_w\sin\theta_e)x_ey_e + v_gy_e\sin\theta_e \\ & + \frac{\omega_g}{K_y}\sin\theta_e - \frac{\sin\theta_e}{K_y}(\omega_g + v_gK_yy_e + v_gK_w\sin\theta_e)\end{aligned}$$

After cancellation of terms:

$$\dot{V} = -K_xx_e^2 + v_gy_e\sin\theta_e - \frac{\sin\theta_e}{K_y}(v_gK_yy_e + v_gK_w\sin\theta_e)$$

After additional cancellation:

$$\dot{\mathbf{V}} = -\mathbf{K_x x_e^2} - \frac{\mathbf{v_g K_w \sin^2\theta_e}}{\mathbf{K_y}} \qquad \mathbf{(16)}$$

Under the condition that the velocity of the tracked object, $\mathbf{v_g}$ is ≥ 0, $\dot{V} \leq 0$ for any error pose, and thus Equation (15) is indeed a Lyapunov function. The implication then, is that for the following conditions, (i) $\mathbf{v_g}$ positive; (ii) $\mathbf{v_g}$ and $\boldsymbol{\omega_g}$ continuous; (iii) $\mathbf{v_g}$, $\boldsymbol{\omega_g}$, $\mathbf{K_x}$ and $\mathbf{K_w}$ reasonably bounded; (iv) accelerations, $\dot{\boldsymbol{v}}_{\mathbf{g}}$ and $\dot{\boldsymbol{\omega}}_{\mathbf{g}}$, are sufficiently small, the error pose, $\mathbf{E_e}$, is asymptotically stable in the manner of Lyapunov under the control rule of (14) [7] [8].

Applying the control law, (14), to the DDMR platform results in the block diagram for the complete system as shown in Figure 4-2. It remains necessary to tune $\mathbf{K_x}$, $\mathbf{K_y}$, and $\mathbf{K_w}$ for proper response. That is accomplished in Chapter 5.

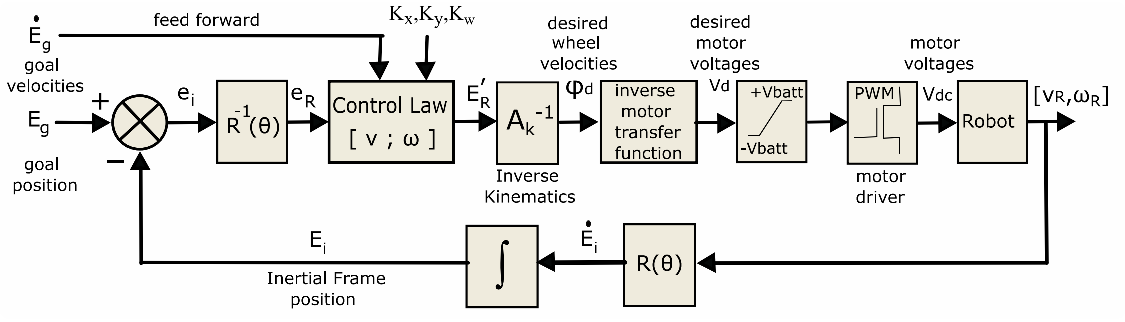


FIGURE 4-2: Control Block Diagram of the DDMR

Chapter 5

Simulation and Tuning Results

In order to examine the impact of tuning the control constants, the robotic platform and its controller are modeled in MATLAB®. The code is consistent with the control block diagram of Figure 4-2. The state-space representation of the DDMR plant is presented in (9f). This set of differential equations is modeled as a set of difference equations with appropriate time-step (**0.1-msec**). Equation (10) is also modeled as a difference equation to map the robot pose to the inertial frame. Equations (12) and (14) are coded into the model to calculate the pose error, translate it to the robot frame and apply it to the control law.

It is also necessary to transform the output of the control law, the desired robot velocity pose $\mathbf{\dot{E}_R = [v, 0, \omega]^T}$, into desired wheel velocities, $\boldsymbol{\varphi_1}$ and $\boldsymbol{\varphi_2}$. This is accomplished using the inverse of Equation (10), finding desired $\boldsymbol{\varphi}$ values when desired $\mathbf{\dot{E}_R}$ is known. At this point, another transformation is necessary, turning desired $\boldsymbol{\varphi}$ into desired motor voltages, $\mathbf{V_{dc1}}$ and $\mathbf{V_{dc2}}$. Equation (9b) describes the relationship between $\boldsymbol{\varphi}$ and $\mathbf{V_{dc}}$. It is not convenient however, to obtain a value for the motor armature currents, **Ia,** or their derivatives from the physical robot. It would require at the least a pair of resistive shunts and 2 ADC conversions during each control loop, adding a significant computational burden to the 8-bit microprocessor. Other current sensing options are available at a price and still add significantly to the computational cost. The proposed solution is a simple linear approximation of (9b) that accomplishes the transformation of **$\varphi \rightarrow$ Vdc** and is implemented in both MATLAB and the microprocessor as Equation (17). **GR** is the gear ratio between motor shaft and wheels.

$$\mathbf{Vdc\ (desired) = GR\ Ka\ \varphi\ (desired)} \qquad \textbf{(17)}$$

Another requirement of the MATLAB model is to account for variations in battery health. Electric torque in the motor is proportional to the square of the armature current, which is in turn dependent on applied voltage from the battery. Therefore, a battery model is developed to mimic the physical battery. Two quantities must be calculated: **AH**, the remaining ampere-hours in the battery, and **Vb**, the battery voltage. The **AH** calculation involves a simple trapezoidal integration of the two motor currents, interpolating between the time-steps and subtracting the area under the current curves, converted to amp-hours, from the existing battery capacity. **Vb** is dependent on **AH** and a generic battery equation from The MathWorks is used as a proxy for alkaline battery performance, avoiding the need to field test actual batteries. The equation is modified slightly for ease of use within an iterative loop and shown below [9]:

$$\mathbf{AH_{new} = AH_{old} - dt(Area\ under\ I_{a1} + Area\ under\ I_{a2})/3600} \qquad \textbf{(18)}$$

$$\mathbf{V_{b\text{-}new} = V_{b\text{-}old} - 0.72(AH_{old} - AH_{new}) - R_b(I_{a1} + I_{a2})} \qquad \textbf{(19)}$$

In many cases, it is necessary to also model the PWM motor driver circuit, since there is a measurable delay in the dynamic output response of a bridge driver circuit. That is not deemed necessary in this case, however. On the physical robot, the rise time for a 100% step change in output voltage (the worst case) is measured at 0.63 milliseconds. The control loop on the robot updates in a slow 10 milliseconds due to a restriction imposed by shaft encoders (discussed in Chapter 6). Therefore, the worst-case delay amounts to 6.3% of the control loop and is not considered significant. Future development of the simulator should include such a model, however, since it is quite common to be running control loops at less than a millisecond .

A flowchart of the MATLAB simulation code is shown in Figure 5-1. An example of the coding structure is depicted as follows. The two state equations describing the armature current, $\mathbf{I_a}$, and shaft velocity, $\boldsymbol{\varphi}$, of the drive motor as it works against the inertia of the robot are coded as difference equations and calculated iteratively. The difference equations for $\boldsymbol{\varphi}$ and $\mathbf{I_a}$ are shown here, with **dt** as the 0.0001 second time-step.

$$\text{Phi(i+1) = Phi(i) + dt*(1/ }J_{m1}\text{)(}F_1\text{*Phi(i) + Kt.*Ia(:,i)) ;}$$

$$\text{Ia(i+1) = Ia(i) + dt*(1/La)*(-Ka.*Phi(i) - Ra.*Ia(i) + Vdc(i+1)) ;}$$

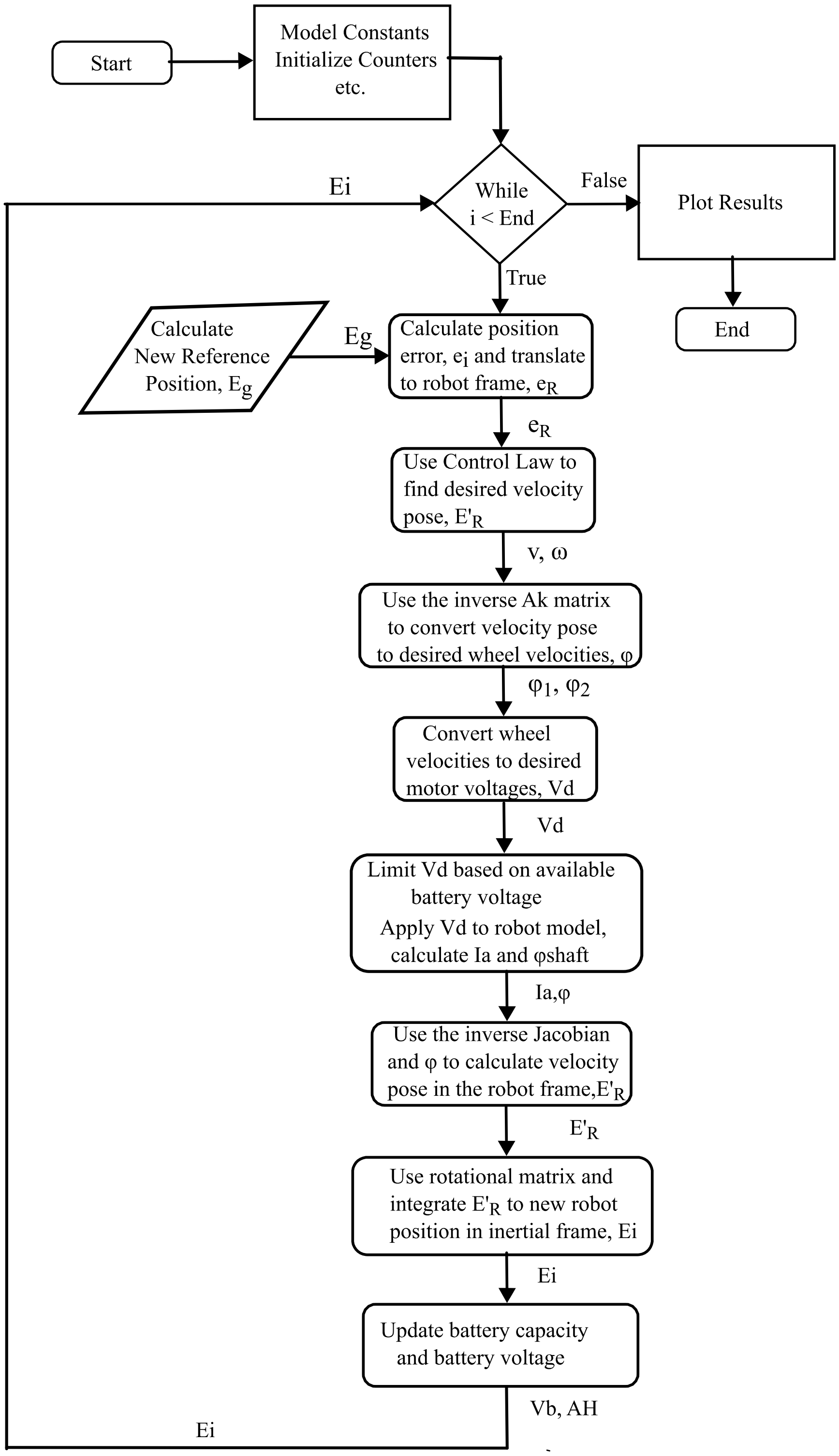


FIGURE 5-1: MATLAB Simulation Flowchart

The completed MATLAB simulator contains physical parameters measured on the actual robot and is useful for fine tuning control response. Since it accurately models the physical system, the numeric values of control parameters determined in the simulator are the same values used in the robot microcontroller code. As described in Chapter 4, the control algorithm has five control parameters, two of which, $\mathbf{v_g}$ and $\boldsymbol{\omega_g}$, are time dependent variables, acquired by the robot while tracking the target object with sensors. Refer to Figure 4-1 for details. It is demonstrated, also in Chapter 4, that the remaining three parameters, $\mathbf{K_x}$, $\mathbf{K_y}$ and $\mathbf{K_w}$, are constants and produce an asymptotically stable system for any combination of positive values (within a reasonable bound). There remains, however, the question of tuning these parameters for a responsive system that provides adequate damping for oscillatory behaviors.

The impact and sensitivity of each parameter, $\mathbf{K_x}$, $\mathbf{K_y}$ and $\mathbf{K_w}$ is first assessed independently. Step-Input simulations are performed on linear paths and circular paths. The influence of each variable is charted for a gain of 0 through 10. The results can be found in Appendix A2. Some pertinent graphs from the sensitivity study start with a line tracking simulation in Figure 5-2, where a step change in X-axis error in the robot frame is simulated. Gain constants $\mathbf{K_y}$ and $\mathbf{K_w}$ are turned to zero, the $\boldsymbol{\omega_g}$ reference is zero for line following and the reference $\mathbf{v_g}$ is set to 0.5 m/sec which is the velocity of the line being tracked.

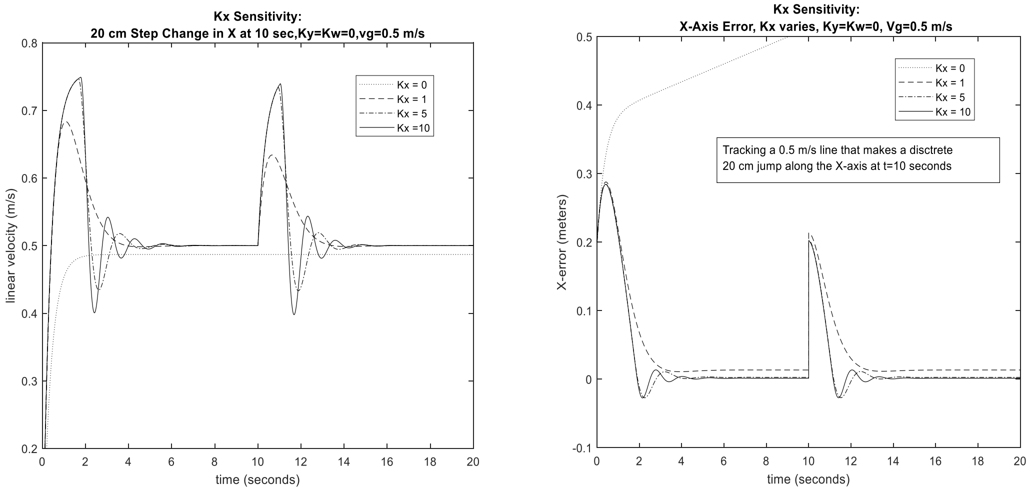


FIGURE 5-2: K_x Sensitivity to a 20cm Step Change

Examining the $\mathbf{K_x}$ sensitivity in Figure 5-2, and inspecting the control law in Equation (14), it is clear that $\mathbf{K_x}$ is a significant component of control scheme responsiveness. As a proportional gain, it is the major driver of change in forward velocity and vital to reducing error in the robot X-direction, $\mathbf{X_R}$. Given a high gain in $\mathbf{K_x}$, Figure 5-2 hints that robot behavior can become oscillatory. However, one can also observe from the linear velocity graph on the left that there is almost no difference in response between curves for $K_x = 5$ and $K_x = 10$. The performance constraints represented by the physical robot tend to limit the effectiveness of gains beyond $K_x = 5$.

Likewise, the gain constant $\mathbf{K_y}$ has a similar effect on error appearing on the Y-axis in the robot frame. Figure 5-3 depicts robot response during line tracking to a 20-centimeter step change in Y-axis error. The sensitivity of behavior to $\mathbf{K_y}$ gain is studied with $\mathbf{K_x} = \mathbf{K_w} = \boldsymbol{\omega_g} = 0$, and $\mathbf{v_g}$ set to 0.5 m/s.

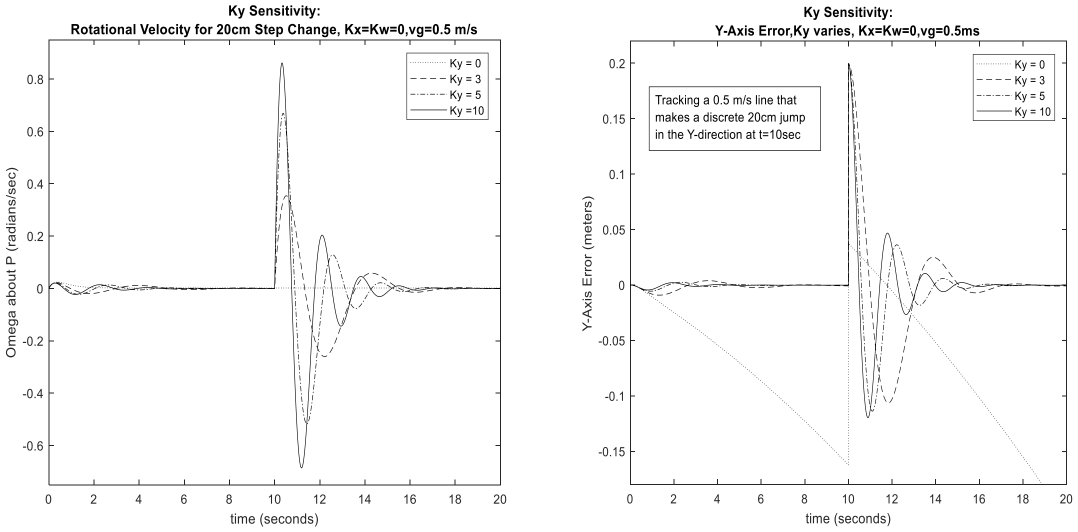


FIGURE 5-3: K$_y$ Sensitivity to a 20 cm Step Change

The examination of $\mathbf{K_y}$ sensitivity indicates that this gain is capable of driving error in the Y-axis to zero in the robot frame and controlling angular velocity. Like K_x, the performance limits of the physical robot limit the effectiveness of gain values beyond $K_y = 10$.

The gain constant $\mathbf{K_w}$ has a different impact on robot behavior. Like $\mathbf{K_y}$, it resides in the angular velocity, $\boldsymbol{\omega}$, equation of the controller but has minor impact on Y-Axis error. In addition, it has an indirect influence on the linear velocity. Looking at Equation (14), $\mathbf{K_w}$ is moderated by $\mathbf{sin\Theta_e}$**,** while $\mathbf{cos\Theta_e}$ moderates reference velocity, $\mathbf{v_g}$, in the linear velocity equation, v. Thus, as $\mathbf{\Theta_e}$ approaches pi/2, $\mathbf{K_w}$ dominates and as $\mathbf{\Theta_e}$ approaches 0, $\mathbf{v_g}$ prevails. The result is that $\mathbf{K_w}$ works as damping factor for both **v** and $\boldsymbol{\omega}$ as shown in Figure 5-4. Notice that $\mathbf{K_w}$ has zero effect on either the angular rotation or the Y-axis error for the step change at 10 seconds. The benefit of $\mathbf{K_w}$ can be seen later in this Chapter, in Figures 5-5 and 5-6.

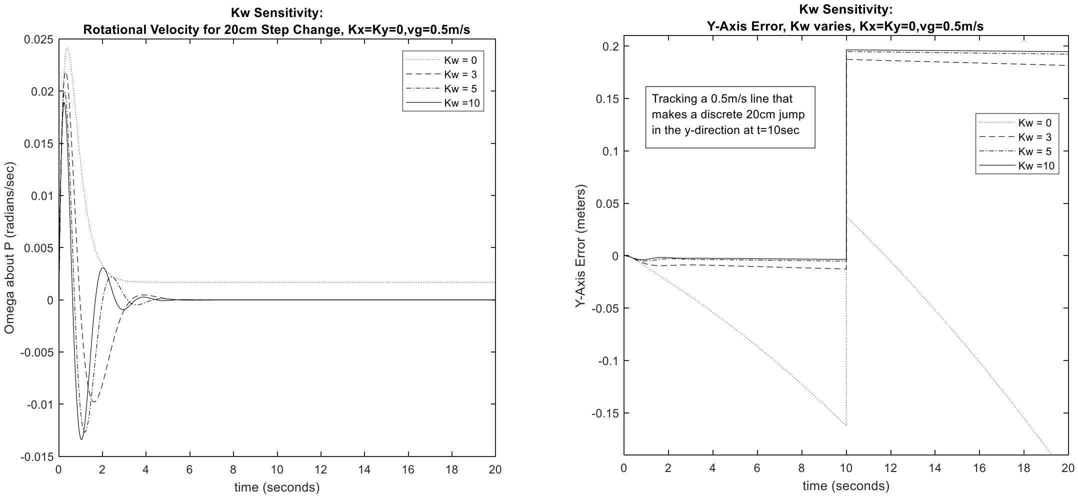
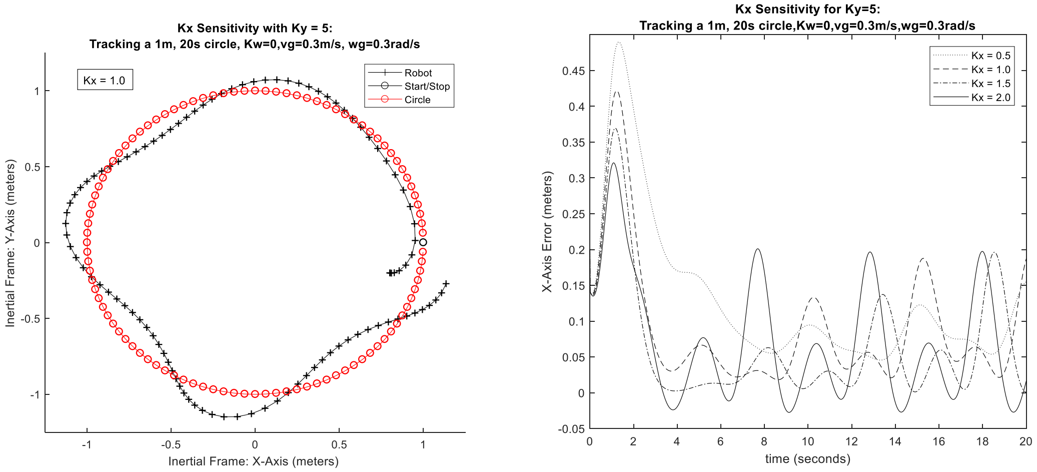


FIGURE 5-4: $\mathbf{K_w}$ Sensitivity to a 20cm Step Change

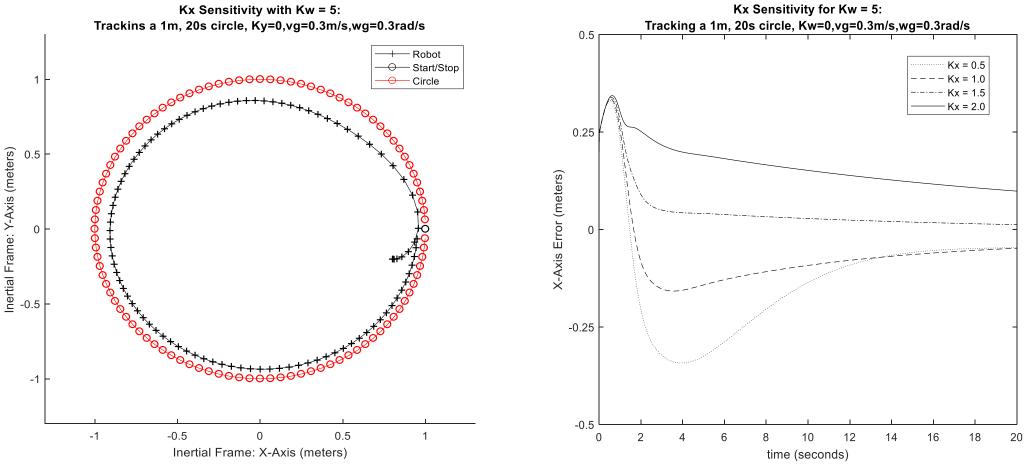
Another factor in determining $\mathbf{K_x}$ and $\mathbf{K_y}$ values is the traction constraint of the robotic platform to which it is applied. Wheel slip is an unacceptable consequence of gains that are set too high, resulting in corruption of wheel encoder feedback. For the robot under study, angular accelerations in the range of 5 rad s^{-2} are sufficient to cause slip, depending upon the road surface.

It is ultimately desired to track an object moving in a uniform circle at a uniform velocity. Tuning should start with $\mathbf{K_x}$. It is important to set it high enough to reap the benefit of its error correction, but it must be bounded by the risk of wheel slip. Careful examination of the slopes of the velocity curves in Figure 5-2 (Kx Sensitivity), indicates that they all have accelerations of less than 0.5 m/s^2. That is a linear acceleration, however. It is important to also consider the outside wheel acceleration during aggressive differential steering. A $\mathbf{K_x}$ value of 1 to 2 is a safe start for the robot under study, where the gain is high enough to be responsive, but not a danger for wheel slip and allowing for meaningful increase in $\mathbf{K_y}$. A sensitivity is now run for $\mathbf{K_x}$ values from 0.5 to 2, first with $K_y = 5$ and $K_w = 0$, then with $K_y = 0$ and $K_w = 5$. Figure 5-5 shows the contribution of $\mathbf{K_y}$ as $\mathbf{K_x}$ varies and Figure 5-6 the impact of $\mathbf{K_w}$ as $\mathbf{K_x}$ varies. Note the distinct difference in robot response to $\mathbf{K_y}$ versus $\mathbf{K_w}$. $\mathbf{K_y}$ aids in the reduction X-axis error in the robot frame (at the expense of damping), whereas $\mathbf{K_w}$ does much less.



(a) Tracking for Ky = 5, Kx = 1 **(b) Error for Ky = 5, Kx varies from 0.5 to 2**

FIGURE 5-5: Impact of $K_y = 5$ as K_x Varies



(a) Tracking for $K_w = 5$, $K_x = 1$ **(b) Error for $K_w = 5$, K_x varies from 0.5 to 2**

FIGURE 5-6: Impact of $K_w = 5$ as K_x Varies

Note, indeed the oscillations for the $K_y = 5$ case, whereas the $K_w = 5$ case is heavily damped. The true benefit of $\mathbf{K_w}$ is as a damper. It appears quite possible to lean heavily upon the damping contribution from $\mathbf{K_w}$, allowing $\mathbf{K_x}$ and $\mathbf{K_y}$ to be set rather aggressively to reduce X and Y errors, respectively. The limit to the aggression appears to lie solely with the physical limits of the robot, especially the angular acceleration of the wheels and its potential for wheel slip.

Now, it is possible to test all three parameters at once. Choosing a value of 1 for $\mathbf{K_x}$, to have some safety margin on acceleration, and setting a value of 6 for $\mathbf{K_y}$ to encourage response the Y-error, a sensitivity on the damping contribution from $\mathbf{K_w}$ is tested. Again, a 1-meter, 20 second circle is tracked. The results are shown in Figure 5-7.

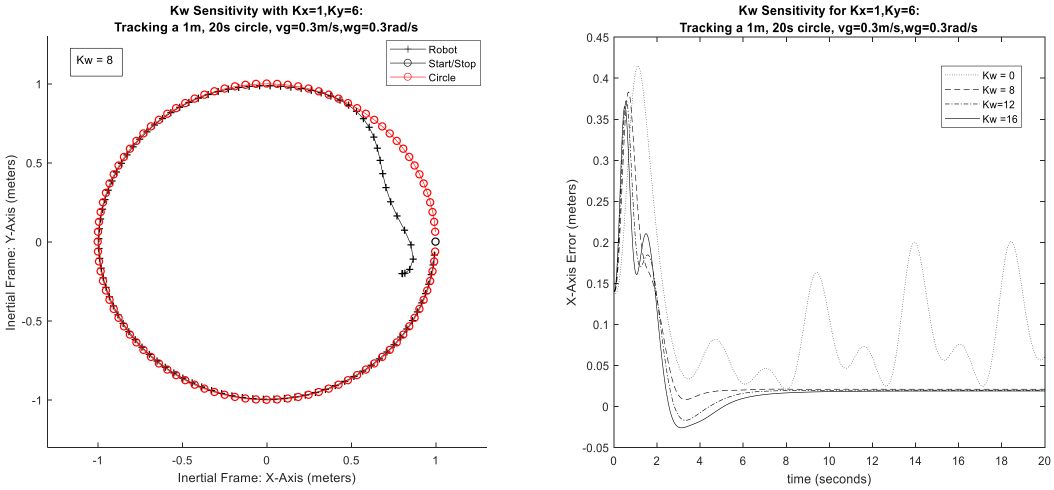


FIGURE 5-7: K_w Sensitivity with K_x = 1, K_y = 6

The value of K_w = 8 appears to provide a response that is just slightly underdamped. Based on the poor start, the **K_x** setting is probably not quite high enough. It remains to be seen how unique these settings are, and how robust. The following is a simple check on uniqueness and robustness. In Figure 5-8, more aggressive settings are used to track two circles, one 40 cm in radius covered in 8 seconds, the other 60cm in 12 seconds. The settings are K_x = 2, K_y = 12 and K_w = 18.

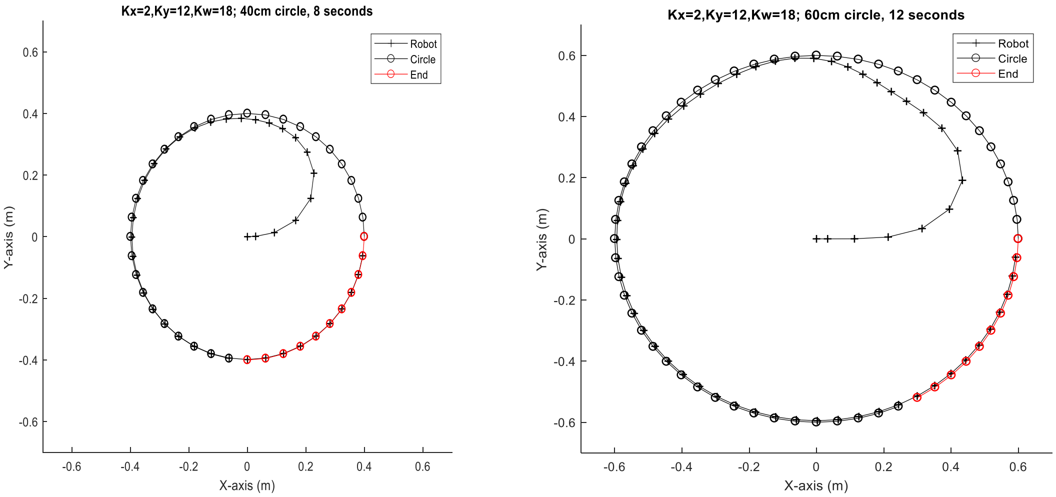


FIGURE 5-8: Robustness Test of Aggressive Controller Settings

Note that these aggressive settings work better on the smaller, faster circle. In the 0.6m circle, the robot initially accelerates too fast, then oversteers in the process of catching up with the object moving on the circle. Testing to date indicates that a diverse combination of the three gain variables will produce comparable performance. For example, the settings $K_x = 1.1$, $K_y = 6$ and $K_w = 8$ will also produce good tracking, perhaps better than the settings shown above. Unfortunately, a thorough examination of the setting interactions, a precise metric for comparison of settings, and the development of a tuning algorithm are beyond the scope of this document.

Chapter 6

The Robot Prototype

The physical robot is deliberately assembled from inexpensive parts**,** including a cut sheet of 3/16" acrylic from an older robot serving as the base. Most of the other parts were purchased from an online hobbyist store, www.sparkfun.com. The total cost is **$90** and a breakdown is presented as Table 1.

TABLE 1: DDMR Component Costs

Robot Component	Unit Price	Qty	Ext. Cost
Gear Motors, wheels, hall encoders	$ 15.95	1	$ 15.95
Arduino Uno (8-bit AVR)	$ 25.00	1	$ 25.00
Dual Motor Driver Shield	$ 19.95	1	$ 19.95
Bluetooth HC-05 Card	$ 9.00	1	$ 9.00
Batteries and Holders	$ 10.00	1	$ 10.00
Misc. wires and hardware	$ 10.00	1	$ 10.00
		Total:	**$ 89.90**

The parts are assembled with some care to place the center of mass just behind the center of rotation, **P**. This allows the robot to rest lightly on a wire skid as the third point of contact. The finished robot is shown in Figure 6-1.

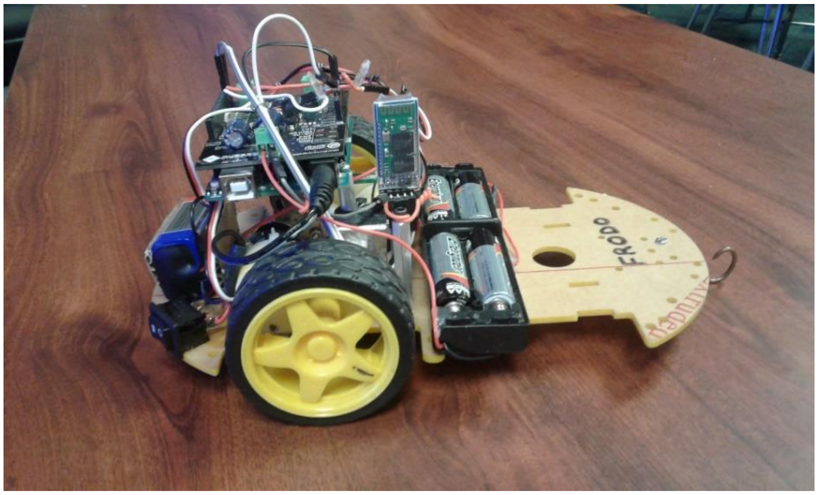


FIGURE 6-1: The Assembled DDMR

The completed robot has an electrical design that is summarized in Figures 6-2 and 6-3. Several physical measurements are necessarily taken from the robot to build an accurate simulation model.

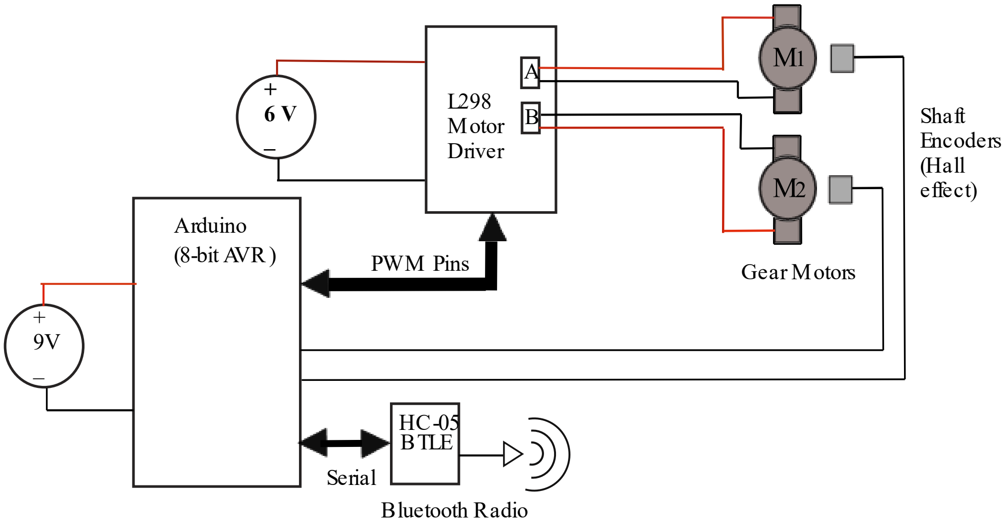


FIGURE 6-2: Electrical Block Diagram of the DDMR

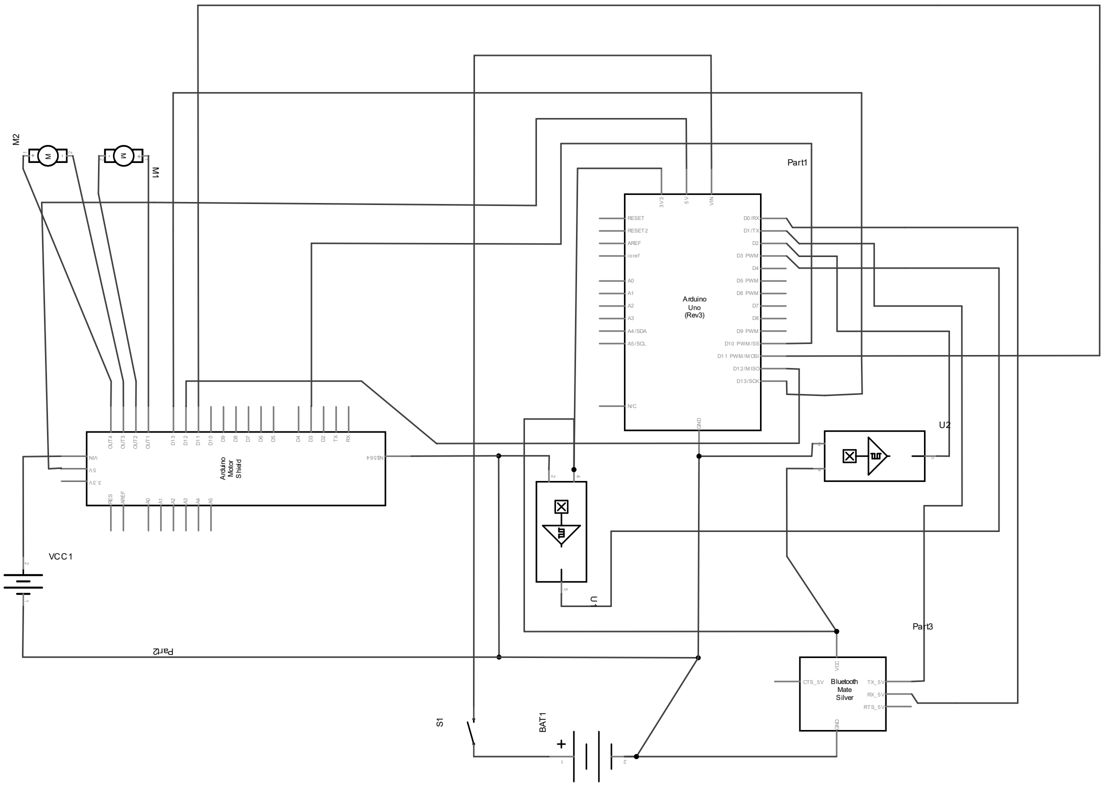


FIGURE 6-3: Electrical Schematic of the DDMR

Table 2 lists the necessary physical parameters of the robot measured for use in the simulation. Measurement equipment includes an ohmmeter, an L-C meter, a hand-held tachometer, a tape measure, a set of calipers and a gram scale.

TABLE 2: Physical Measurements of the DDMR

Parameter	Measurement Method	Value
Mass, m	gram scale	405 grams
Wheel Radius, ($r_1 = r_2$)	caliper	32.2 mm
Wheel Offset, L	tape measure	64 mm
Armature Resistance, R_{a1}	ohm-meter	5.32 ohms
Armature Resistance, R_{a1}	ohm-meter	5.41 ohms
Armature Reactance, L_{a1}	L-C meter	0.023 H
Armature Reactance, L_{a2}	L-C meter	0.024 H
Wheel RPM, No Load	tachometer	210 RPM
Gear Ratio, GR	Observed	48:1

It is also necessary to make a reasonable estimation of the moment of inertia of the robot as experienced by each drive motor, and estimate the frictional forces impeding wheel motion. Starting with the equation, $\boldsymbol{\Sigma\ \tau = J\ \alpha}$, it is possible to make a crude estimate of the moment:

$$\tau_{\text{motors}} = Fd = Fr = (ma)r = J\alpha = Ja\ /\ r\ ; \quad mar = Ja\ /\ r\ ; \quad \mathbf{J = mr^2} \qquad \mathbf{(20)}$$

$$\text{For the DDMR,}\ \ J = 0.405(0.0322)^2 = \mathbf{0.0004\ kg\ m^2}$$

The drive motors, since they are rotational, see the robotic platform as an inertial load, especially when it is moving in purely linear motion. As indicated in comparing Equation (7) to Equation (8), there is potential for additional torques in the DDMR, especially related to a moment around the point, P. Again, if the acceleration of the system is held to reasonable values, these additional forces are small and therefore ignored. The frictional forces, **F**, are difficult to measure. It is commonly assumed that they are about 10% the size of the inertial moment, **F = 0.1J**. It can commonly be disproportionately high on small robots, but can be fine-tuned when comparing simulated behavior to the actual robot.

The robot uses an Arduino Uno using the Atmel Atmega328 MCU, a member of their 8-bit AVR series architecture, with 8 k-byte program memory and a 16 MHz clock. Programming the Arduino for the tracking controller described uses approximately 30% of memory and runs a typical control loop in about 0.8 milliseconds.

However, there is a limiting factor in loop speed: The motor shafts are set up with simple encoder disks containing 4 magnets and a Hall effect sensor. Each Hall sensor is wired to an interrupt pin on the Arduino, set to trigger an interrupt at every change in logic level (both rise and fall). The interrupt service routine increments a counter for each motor/wheel assembly. Thus, there are 8 pulses counted for each motor shaft revolution and considering the 48:1 gear ratio, 48*8 or 384 pulses per wheel revolution. With a wheel radius of 32.2 mm, the linear distance of one revolution is 2pi*32.2/384 = 0.52 mm/pulse. While this would appear to be reasonable resolution, it is necessary to consider that even at no load speed (210 RPM) and a loop speed of 0.8-ms, the result is:

$$\frac{(\mathbf{210\ rev/min})}{(\mathbf{60\ sec/min})}(\mathbf{384\ pulses/rev})(\mathbf{0.0008\ sec/loop}) = \mathbf{1.08\ pulses/loop} \tag{21}$$

Thus, during a typical pass through the 0.8-ms control loop, each encoder will record either 1 or zero pulses. This is an unacceptably coarse resolution. Therefore, the control loop is artificially lengthened with 9-ms of delay. At 10-ms per loop, the resolution at no load speed is 13.4 pulses per loop. While this is not ideal, it is likely that any additional length in the control loop will degrade control response and ultimately, controllability.

A Bluetooth LE module, HC-05, is also interfaced with the serial pins on the Arduino. This allows wireless telemetry between the robot and another BTLE device, such as a PC or smart phone. Wireless commands such as Start/Stop, may be sent to the robot. In addition, the robot can send back information during or after a test run.

Chapter 7

Prototype Results versus Simulation

The data gathered from multiple tests verify the ability of the simulator to accurately model the actions of the physical robot. Shown here are summaries of two tests, one linear and one circular.

The linear test is performed open loop, with no feedback control. The robot is programed to incrementally apply equal voltages to the drive motors, gradually accelerating the machine to full speed over the course of 5 seconds. The trajectory is reasonably straight, although slight differences in the motor impedances and irregularities in the tires prevent the robot from taking a straight course in open loop operation. Likewise, the simulator is programmed to match the incremental application of motor voltages, and since it models the differences in motor impedances, it too drifts slightly away from a straight line. Shown below are two graphs comparing the performance of the simulation to the robot for the linear test. The left-hand graph of Figure 7-1 directly compares the distance traveled by the two, plotted against run-time while the right side plots the error, $X_{simulation}$ - X_{robot}, also as a function of time. There are a host of minor non-linearities in a small, inexpensive robot, including gear lash, axle wobble, tire deformity and intermittent frictional forces. Few of these are modeled in the simulator and none to any fine degree.

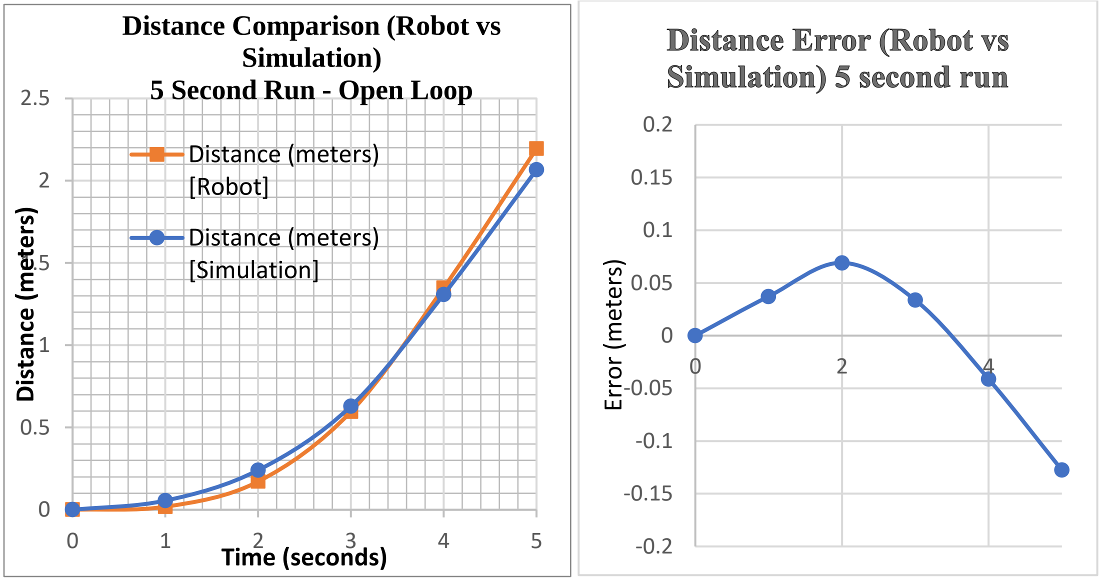


FIGURE 7-1: Linear Distance Test – Simulation versus Robot

Another test demonstrates the closed loop behavior of the robot and the corresponding simulation. With the feedback controller fully implemented and control gains set at $K_x = 1.1$, $K_y = 6$, and $K_w = 8$, both entities, are programmed to read data from an imaginary tracked object. The data describes a 0.6-meter circle that the object traverses, completing a full revolution in 12 seconds. This corresponds to the object traveling at a linear velocity of 0.314 m/s and an angular velocity of 0.524 rad/sec. These velocities can be programmed into both the simulator and the robot. It is a tribute to the robustness of the controller and stability of the physical robot that these observed values need not be particularly accurate for proper tracking. The gain constants adapt quite well to observation error. The robot or simulator is placed at the origin of the circle and begins to catch up to the moving fictitious object. The results are shown in Figure 7-2 and 7-3 below:

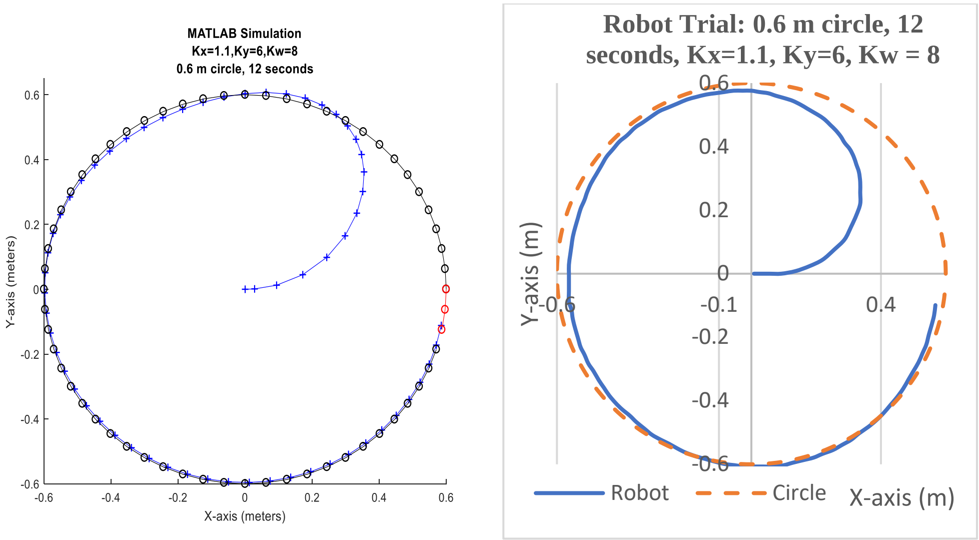


FIGURE 7-2: Circle Test: Closed Loop, Robot vs Simulation, Tracking an Object moving in a Circle

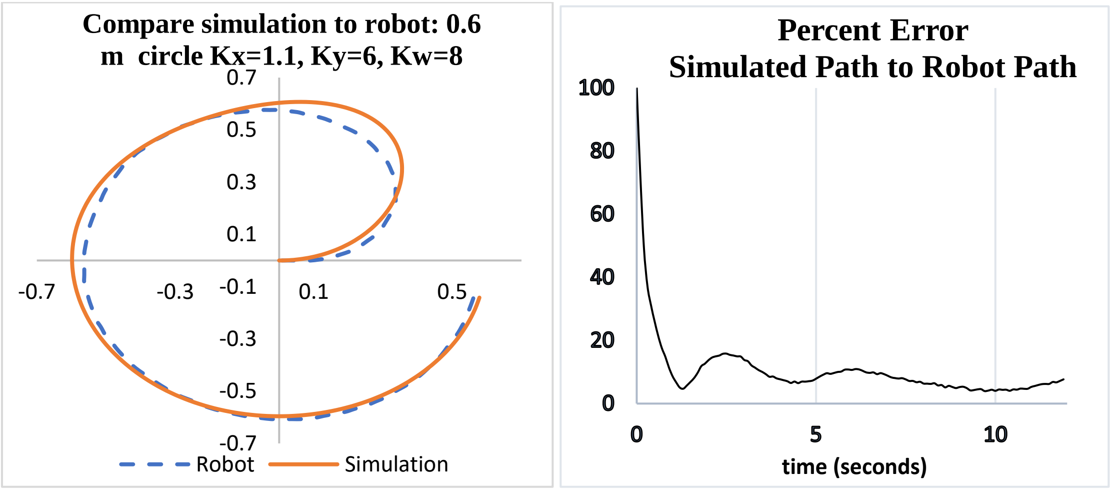
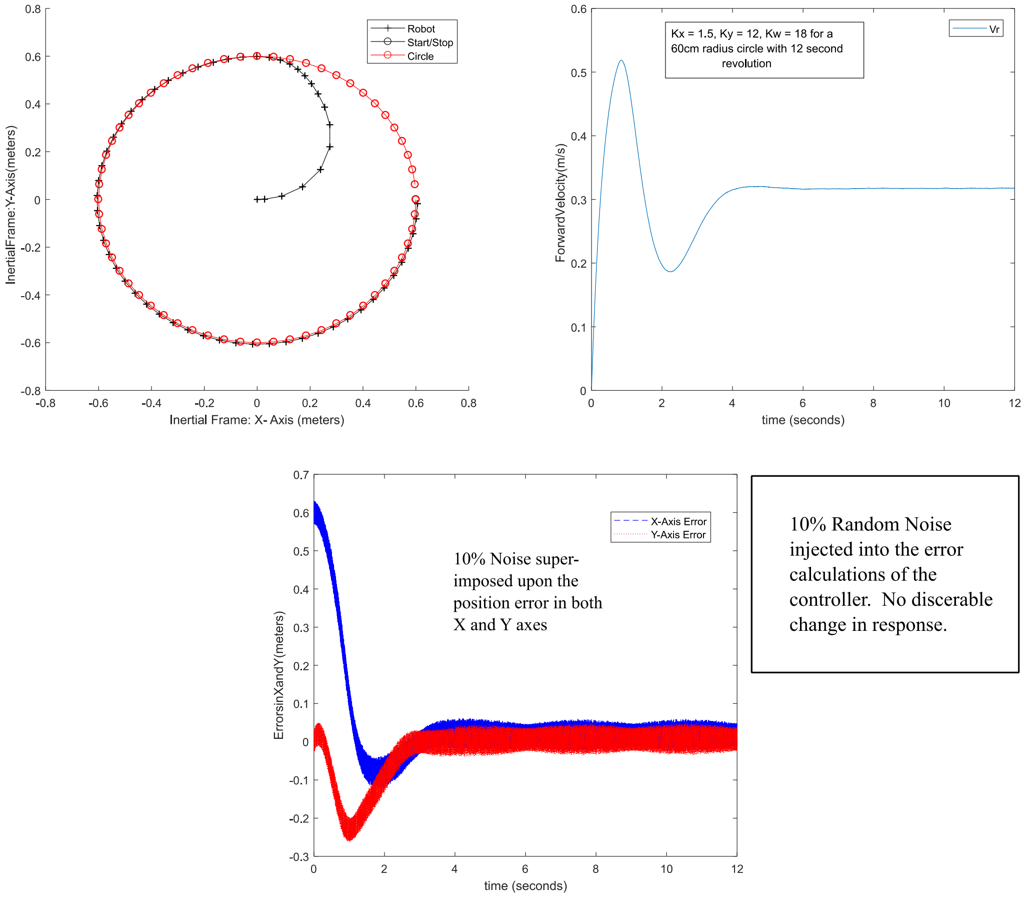


FIGURE 7-3: Additional Closed Loop Results

While the results aren’t perfect, they are quite encouraging. In the linear run, the simulation predicted an endpoint with an error of 12.8 cm from actual, over a run of 2.19 meters (5.8% error). The circle test, with closed loop control, should prove even better. Over a perimeter length of 3.77 meters, the simulator predicted an endpoint of [0.585, -0.112] while the robot ended at [0.568, -0.099]. Treating each as a vector from the origin, the norm of the simulator vector is 0.596 meters, while the norm of the robot vector is 0.577. The norm of the error between the two vectors is 0.021. The ratio of the error norm to the norm of the robot vector is 3.7%, a modest improvement over the open loop results.

The robot and simulator are also tested on 0.4 meter, 8 second course using the same gain constants of Kx=1.1, Ky = 6 and Kw=8. The results are similar although, at 0.4-meter radius and 8 second period, the observed goal velocities of the tracked object are 0.314 m/s and 0.785 rad/sec. The robot would benefit from a higher gain. In addition, a small sensitivity trial was performed with random measurement error. All other test runs were performed with ideal tracking data. The following plots show the results of injecting 10% random noise to the tracking error. There is no measurable difference in robot performance in the simulator. The physical robot was not tested with random noise injection. In any event, the side-by-side comparisons of Figure 7-4 show great promise. The simulator demonstrates enough accuracy to justify its use in rapid prototyping of gain constants for the physical robot.



Kx = 1.5, Ky = 12, Kw = 18 for a 60cm radius circle with 12 second revolution

10% Noise super-imposed upon the position error in both X and Y axes

10% Random Noise injected into the error calculations of the controller. No discerable change in response.

FIGURE 7-4: The Control Law Exhibits Insensitivity to Noise

Chapter 8

Conclusion

In this thesis, a fundamental basis for the kinematic description of Differential Drive Mobile Robot (DDMR)s is presented. The dynamics of wheeled robots, actuated by permanent magnet DC motors are developed. The state space of DDMR platforms is explored and found to be non-linear. A non-linear control law, based on Kanayama's conference paper, is developed and determined to be bounded by a Lyapunov function and asymptotically stable. Using MATLAB, the entire closed-loop DDMR system is modeled with time-stepped difference equations. A simple set of physical measurements is presented that provide reasonably accurate model data. Methods for tuning the control gains are explored. A prototype robot is constructed from inexpensive parts, including a very modest 8-bit processor. A significant limitation in the robot is discovered, that being the 384 pulse wheel encoders. Even with the limitation, reasonable correlation between the physical robot and the simulated robot is observed**.** The limitation does not hinder the robot's ability to successfully implement a non-linear control scheme.

The non-linear control law is shown to be robust on an 8-bit, 16 MHz processor, even when the control loop is forced to run at a slow 10 milliseconds due to encoder limitations, and even when 10% random noise is injected as tracking error. That is an unexpected success. While it was not tested, it seems reasonable to expect the control law to remain robust when real sensors generate the tracking data. However, the limited processing power of this platform is not suited to the machine vision sensors necessary to generate tracking data.

The simulator performs very well for prototyping, predicting robot behavior with acceptable precision. The simplification of the dynamics, Equation (8), and the limited amount of physical model data were not overly detrimental to its accuracy. A better model of friction would be useful. In small robots, the friction forces are many, and significantly non-linear. Ultimately, the non-precision parts prevent a truly accurate model from being completed. Gear lash and axle wobble are just two of many uncertainties that come with a low budget platform.

The data presented here underscore the conclusion that modern control schemes are applicable to slow, imprecise hardware. Limited processing power is not a barrier to performance at the small scale, and a few simple measurements are adequate to generate reasonably accurate computer models for inexpensive robotic platforms.

Bibliography

[1] O. Garret, "10 Million Self-Driving Cars Will Hit the Board by 2020," *Forbes Magazine*, 3 March 2017.

[2] R. Bogue, ""Domestic Robots: Has their time finally come?," *Industrial Robot: An International Journal,* vol. 44, no. 2, pp. 129-136, 2017.

[3] P. Petrov, "Control Algorithms for an Autonomous Vehicle," in *Proceedings of the IEEE Intelligent Vehicle Symposium,* Tokyo, 1993.

[4] R. Siegwart, Autonomous Mobile Robots, 2nd Edition, Cambridge, MA: The MIT Press, 2011.

[5] R. Fierro and F. Lewis, "Control of a Nonholonomic Mobile Robot: Backstepping Kinematics into Dynamics," in *Proceedings of the 34th Conference on Decision and Control*, New Orleans, 1995.

[6] B. C. Kuo, Automatic Control Systems, 4th Edition, Englewood Cliffs, NJ: Prentics-Hall, Inc., 1982.

[7] Y. Kanayama, Y. Kimura, F. Miyazaki and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *International Conference on Robotics and Automation*, Cincinnatti OH, 1990, pp. 384-389.

[8] H. K. Khalil, Nonlinear Systems, Upper Saddle River, New Jersey: Prentice-Hall, 1996.

[9] MathWorks, "Battery - generic model," https://www.mathworks.com/help/physmod/sps/powersys/ref/battery.html.

[10] P. Petrov and L. Dimitrov, "Non-Linear Path Control for a Differential Drive Robot," *Recent*, vol. 11, no. 1 (28), pp. 41-45, 2010.

[11] C. Chen, Linear System Theory and Design, 3rd Edition, New York; Oxford: Oxford University Press, 1999.

[12] R. L. S. Sousa, "Trajectory Tracking Control of a Nonholonomic Mobile Robot with Differential Drive," in *ARGENCON*, Buenos Ares, 1-6, June, 2016, pp.1-6.

[13] Y. Yang and J. Du, "A Trajectory Tracking Robust Controller of Surface Vessels with Disturbance Uncertainties," *IEEE Transactions on Control Systems Technology,* vol. 22, no. 4, pp. 1511-1518, 2014.

Appendices

A1 – Simulation in MATLAB

********* Code for a typical .m file in MATLAB *******************************

```
% Diff'tial drive robot, Controls based on Kanayama [6]:
% 2 drive wheels and a castor, Lyapunov Stable
% Tracking a 0.6 m radius, 12 second circle
% Iterative solution of state equations
% Scott R. Norr
%
% Set up iteration counter, timestep (in seconds), real
% time counter (seconds)
  i=1;dt=0.0001;t(1)=0;

% Robotic Platform Constants
% Wheel radii and robot half-dimensions in meters

  r1=.0322;r2=.0322;l=.064; x=0.05;d=0.01;m=0.405;

% Controller Gain Constants
  kx=1.1;ky=6;kw=8;vref=0.3;wref=.3;

% Reference is the Global Frame at [0,0,0],
% Initial position of tracked object is G
  Ref=[0;0;0];G=[0.6;0;pi()/2];

% Initial position vector of Robot, Ei, in the global frame
  Ei(:,1)=[0;0;0];

% Calculate initial error vector:
  e(:,1)=G(:,1)-Ei(:,1);

% A-matrix for differential drive, where
% wheel velocities,[phi]=inv(A)*[v;w]
  A=[r1/2 r2/2;r1/(2*l) -r2/(2*l)];
  Ainv=inv(A);

% Inverse of the Jacobian matrix of rolling and sliding
% constraints on robot motion:
% Where Jac =[Vx, 0, W]
  Jinv=[0.5  0.5  0; 0  0  1; 1/(2*l)  -1/(2*l) 0];

% initialize other state variables, robot velocity
% and rate of rotation wrt global frame
```

```
  v(1)=0;w(1)=0;vdot(1)=0;wdot(1)=0;u=[0;0];

% Phi are the desired wheel velocities, (rad/sec)
  phi(1,1)=0;phi(2,1)=0;

% Motor model initialize (GR is gear ratio of motors),
% two drive motors (each slightly different)
% Jm is effective inertial moment of robot,
% B1,2 dynamic friction
  Ra=[5.32;5.31];La=0.0023;Ka=[.0054;.0054];Kt=Ka;
  GR=48;Jm=.0001/GR;B1=.07*J;B2=B1;La1=[0.0023;0.0026];

% motor variables, VDC1, VDC2 are the voltages
% applied to each motor thru a motor controller,
% Ia are the armature currents, omega are the
% shaft velocities of the motor.  GR will change
% shaft vel. to wheel vel.
  Vdc=[0;0];Omega=[0;0];Ia=[0;0];Iabs=[0;0];E(1)=0;
  Vbatt(1)=6.1;AH(1)=2.5;Rbatt=0.037;%6V alkaline battery
  tau=[0;0];Ides=[0;0];

% While loop to calculate state variables on an
% iterative basis

while i < 120001

% Update the real time counter
    t(i+1)=t(i)+dt;

% Generate line to track: 1) .6m radius circle in 12sec
    G(:,i+1)=G(:,i)+dt*[-.6*pi*sin(pi()*t(i+1)/6)/6;
                       .6*pi()*cos(pi()*t(i+1)/6)/6;
                                pi()/6];

% calculate new error in Robot Frame of reference
    e(:,i+1)=[cos(Ei(3,i)) sin(Ei(3,i)) 0;
             -sin(Ei(3,i)) cos(Ei(3,i)) 0;
                  0            0     1]*[G(:,i+1)-Ei(:,i)];

% control law for velocity and rate of rotation
% about robot center,(Kanayama)
    v(i+1)=vref*cos(e(3,i+1))+kx*e(1,i+1);
    w(i+1)=wref+ky*e(2,i+1)+kw*vref*sin(e(3,i+1));

%Desired angular velocities at the Wheels, phi-1 and phi-2:
    phi(:,i+1)=Ainv*[v(i+1);w(i+1)];
```

```
%Desired Motor Voltages
    Vdc(:,i+1)=GR*Ka.*phi(:,i+1);

%Limit applied Vdc to within battery parameters
    if Vdc(1,i+1) > Vbatt(i)
        Vdc(1,i+1)=Vbatt(i);
    elseif Vdc(1,i+1) < -Vbatt(i)
        Vdc(1,i+1)= -Vbatt(i);
    end
    if Vdc(2,i+1) > Vbatt(i)
        Vdc(2,i+1)=Vbatt(i);
    elseif Vdc(2,i+1) < -Vbatt(i)
        Vdc(2,i+1)= -Vbatt(i);
    end

% Calculate actual motor shaft velocities and
% armature currents
 Omega(:,i+1)=Omega(:,i)+dt*(B1*Omega(:,i)+Kt.*Ia(:,i))/Jm;
 Ia(:,i+1)=Ia(:,i)+dt*(1/La)*(-Ka.*Omega(:,i)-Ra.*Ia(:,i)+Vdc(:,i+1));

% Limit shaft velocities to no load speed if necessary:
    if Omega(1,i+1)> 1200 % 1200 rad/sec at motor shaft
        Omega(1,i+1)= 1200;
    end
    if Omega(2,i+1)> 1200
        Omega(2,i+1)= 1200;
    end

%battery model:  four alkaline batteries in series
    Iabs(:,i+1)=abs(Ia(:,i+1));

%energy equation to compute remaining energy in battery
    AH(i+1)=AH(i)-dt*(Iabs(1,i)+0.5*(Iabs(1,i+1)-
Iabs(1,i)))/3600-dt*(Iabs(1,i)+0.5*(Iabs(1,i+1)-
Iabs(1,i)))/3600;

%Battery model using a curve from panasonic
    Vbatt(i+1)= Vbatt(1)-0.72*(AH(1)-AH(i+1))-
Rbatt*(Iabs(1,i)+Iabs(2,i));

% Calculate the new position vector in global
% frame, with GR correct:
    Ei(:,i+1)=Ei(:,i)+dt*[cos(Ei(3,i)) -sin(Ei(3,i)) 0;
                          sin(Ei(3,i))  cos(Ei(3,i)) 0;
                              0           0
1]*Jinv*[r1*Omega(1,i+1)/GR;r2*Omega(2,i+1)/GR;0];
```

```
    %Plot new robot pose
    if i==1
         %plot the pose of the robot for the first timestep
        P=[Ei(1,i),Ei(2,i)];O=Ei(3,i);
        c1=P+[x*cos(O)+l*sin(O), x*sin(O)-l*cos(O)];
        c2=P+[x*cos(O)-l*sin(O), x*sin(O)+l*cos(O)];
        c3=P+[-x*cos(O)-l*sin(O),-x*sin(O)+l*cos(O)];
        c4=P+[-x*cos(O)+l*sin(O), -x*sin(O)-l*cos(O)];
       M=[c1(1) c1(2);c2(1) c2(2);c3(1) c3(2);c4(1) c4(2)];
        f1=figure;
        h1=animatedline('Marker','+');
        h2=animatedline('Marker','o');
        addpoints(h1,P(1,1),P(1,2));
        addpoints(h2,G(1,i),G(2,i));%Also plot the tracked object Position);
        drawnow
    end
    if i==100000
        h2=animatedline('Marker','o','Color','r');
    end
    if mod(i,2000)==0
         %plot the pose of the robot every 2000 timesteps
        P=[Ei(1,i),Ei(2,i)];O=Ei(3,i);
        c1=P+[x*cos(O)+l*sin(O), x*sin(O)-l*cos(O)];
        c2=P+[x*cos(O)-l*sin(O), x*sin(O)+l*cos(O)];
        c3=P+[-x*cos(O)-l*sin(O),-x*sin(O)+l*cos(O)];
        c4=P+[-x*cos(O)+l*sin(O), -x*sin(O)-l*cos(O)];
        N=[c1(1) c1(2);c2(1) c2(2);c3(1) c3(2);c4(1)
c4(2)];
        M=[M;N];
        addpoints(h1,P(1,1),P(1,2));
        addpoints(h2,G(1,i),G(2,i));% plot the object
        drawnow
    end

   % Increment iteration counter
    i=i+1;
end

%Plot end of run information

%X-Y plot of center of robot motion in global frame
figure('Name','X-Y Plot of Robot Position in Global
Frame');
plot(Ei(1,:),Ei(2,:))
figure('Name','Motor 1 and 2 - Applied Voltages');
plot(t,Vdc(1,:),t,Vdc(2,:))
figure('Name','Motor 1&2 Currents');
```

```
plot(t,Ia(1,:),t,Ia(2,:))
figure('Name','Robot Path in Global Frame');
plot(M(:,1),M(:,2))
figure('Name','Battery voltage');
plot(Vbatt)
%Wheel 1&2 Angular Velocity
figure('Name','Wheel 1&2 Vel. (RPM)');
plot(t,30*Omega(1,:)/(pi()*GR),t,30*Omega(2,:)/(pi()*GR))
%Wheel 1&2 Linear Velocity
figure('Name','Linear (Xr-direction) Vel. (m/sec)');
Vxr=(r1*Omega(1,:)+r2*Omega(2,:))/(2*GR);
plot(t,Vxr)
%X error and Y error
figure('Name','X and Y errors in Global Frame (meters)');
plot(t,e(1,:),t,e(2,:))
```

A2 – Control Variable Sensitivity Studies

The following pages show some of the sensitivity studies performed to determine the impact of each gain constant, Kx, Ky and Kw.

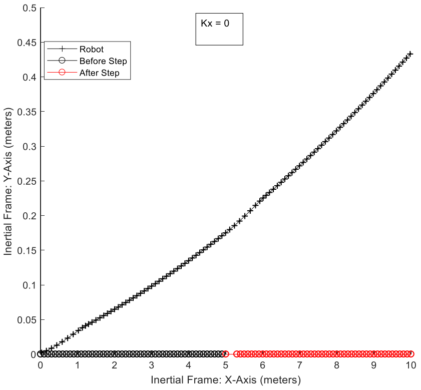
The first group of plots (3 pages) are the results from tracking an object moving along a straight line at 0.5 m/s, with a step change in either the X direction or the Y direction occurring at 10 seconds in the robot frame. Two of the three gain constants are held to zero. The third constant is varied to demonstrate its sole impact on performance.

The second group (6 pages) performs the same type of sensitivity (vary one parameter with the others held to zero) when tracking an object around a 1 meter radius circle. No step change occurs.

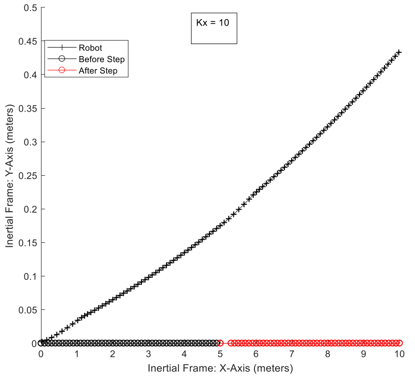
The final two pages are tracking a circle with either Ky = 0 and Kw = 5 while varying Kx, or Ky = 5 and Kw = 0 while varying Kx. This explores how robust the control scheme is with only two control parameters. (Kx is essential for control and is not set to zero in this sensitivity run)

20cm Step in X-Axis: Kx Sensitivity, Ky = Kw=0, vg=0.5m/s, wg=0

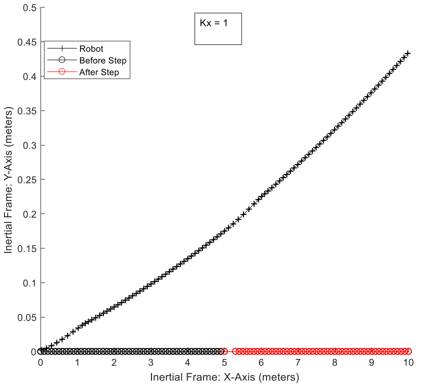
Kx = 0:



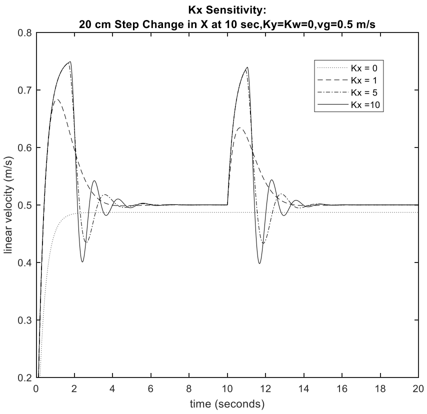
Kx = 10



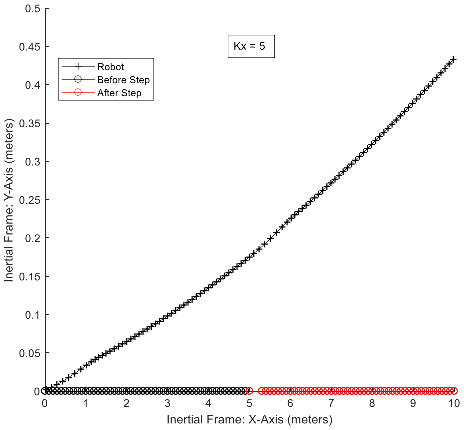
Kx = 1



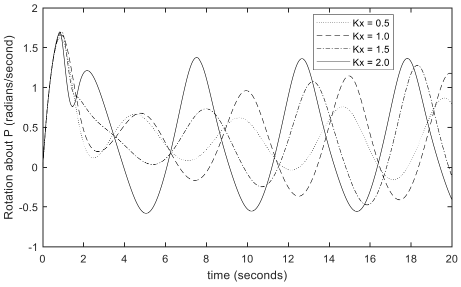
X-Velocity (V_R)



Kx = 5

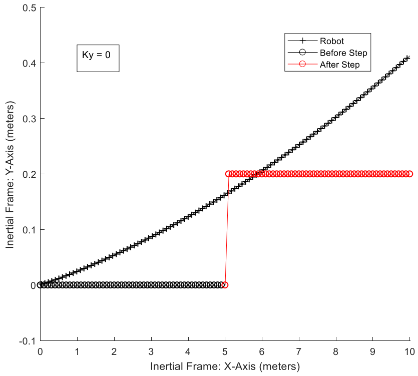


X-Rotation (ω_R)

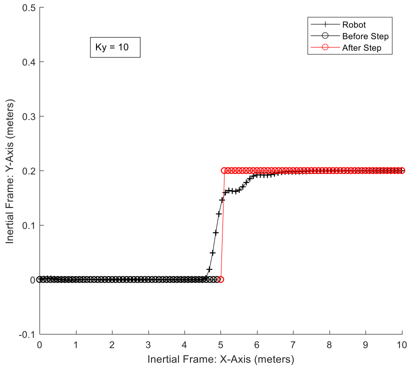


20cm Step in Y-Axis: Ky Sensitivity, Kx = Kw=0, vg=0.5m/s, wg=0

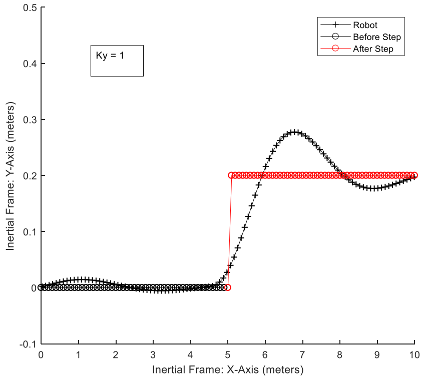
Ky = 0



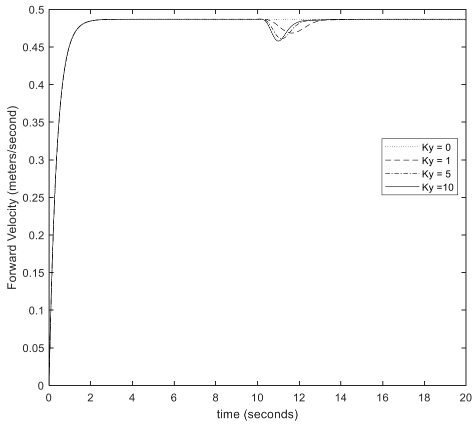
Ky = 10



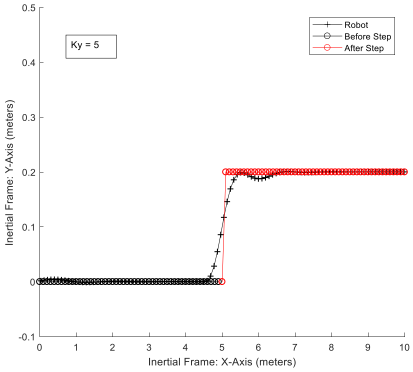
Ky = 1



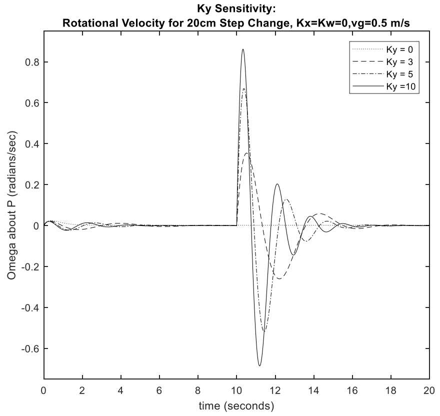
Y-Forward (V_R)



Ky = 5

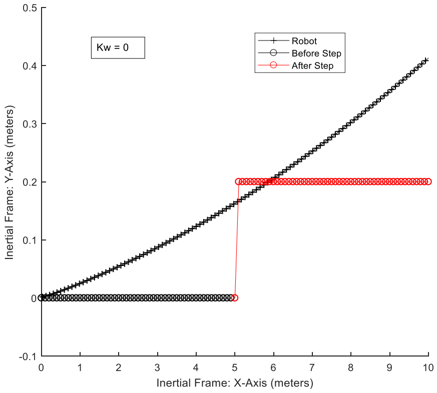


Y – Rotation (ω_R)

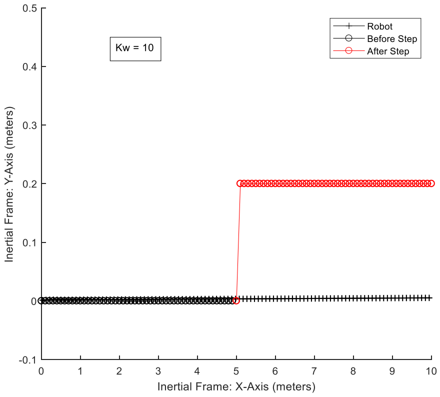


20cm Step in Y-Axis: Kw Sensitivity, Kx = Ky=0, vg=0.5m/s, wg=0

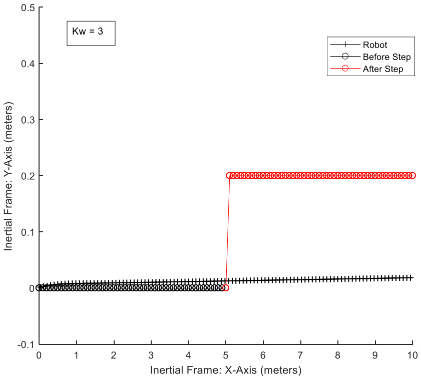
Kw = 0



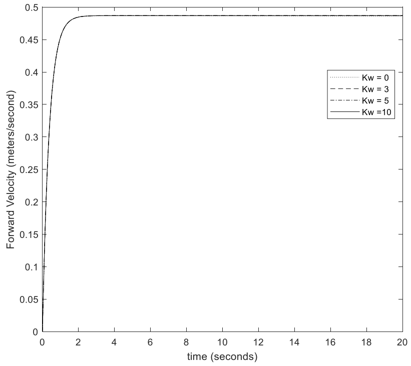
Kw = 10



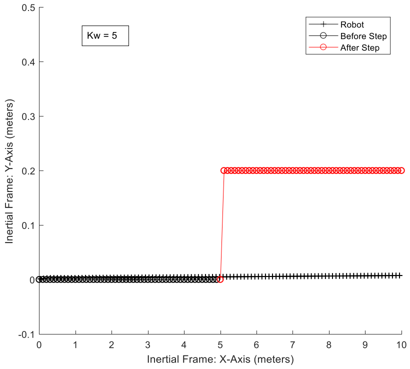
Kw = 3



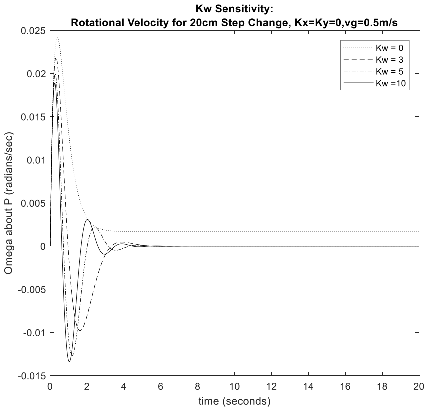
W – Forward (V_R)



Kw = 5

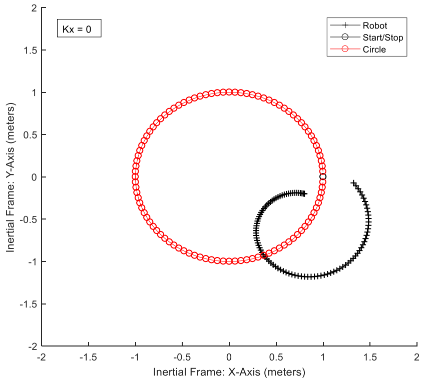


W – Rotation (ω_R)

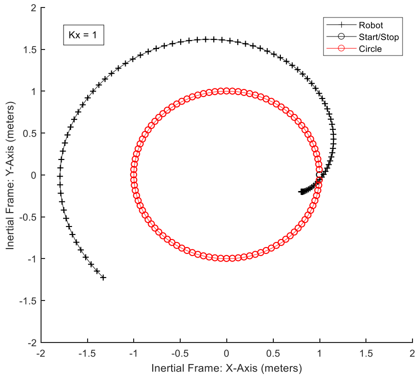


1-meter circle, 20 seconds, Kx Sensitivity, Ky=Kw=0, vg=0.3m/s, wg=0.3rad/s

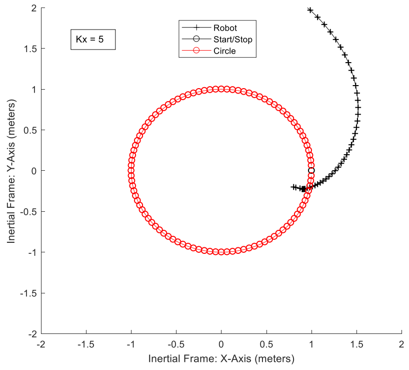
Kx=0



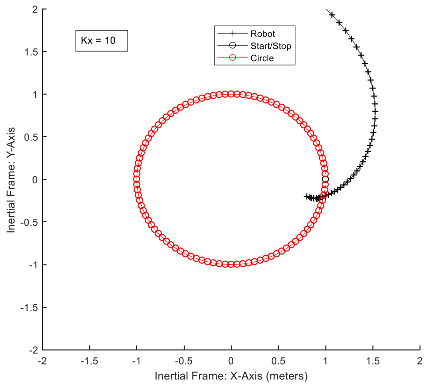
Kx=1



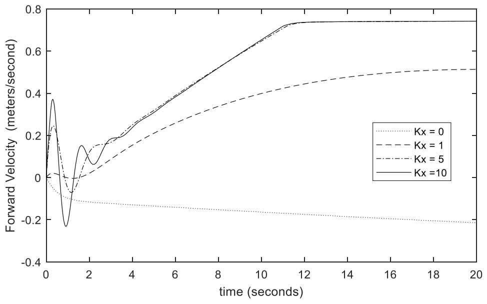
Kx=5



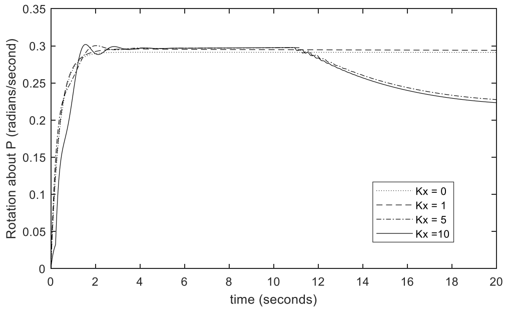
Kx=10



Forward Velocity (v_R)

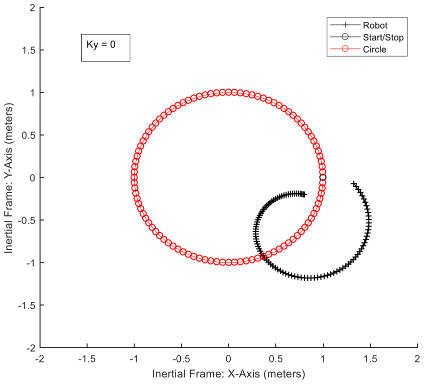


Rotational Velocity (ω_R)

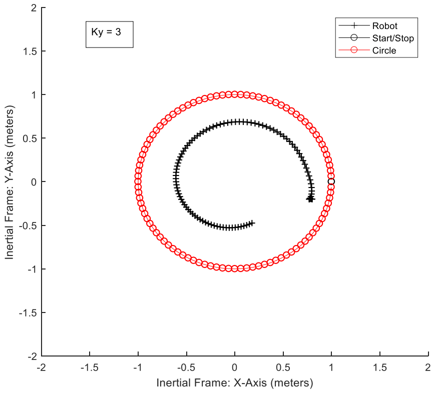


1-meter circle, 20 seconds, Ky Sensitivity, Kx=Kw=0, vg=0.3m/s, wg=0.3rad/s

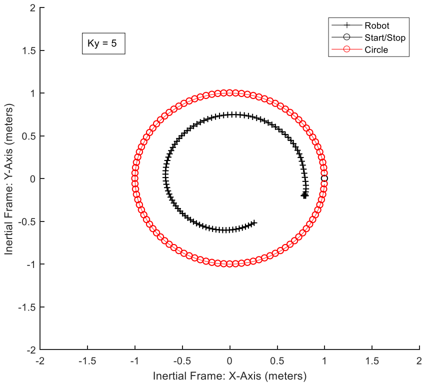
Ky=0:



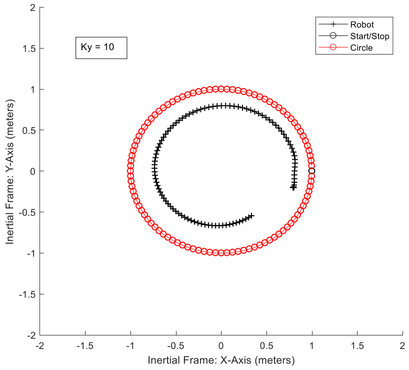
Ky=3:



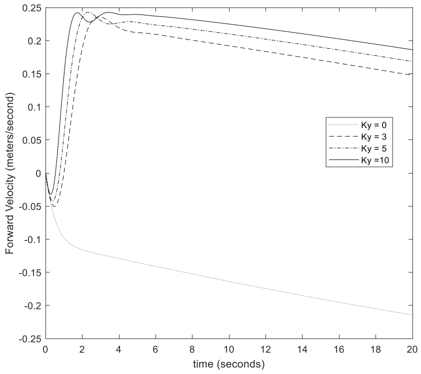
Ky=5:



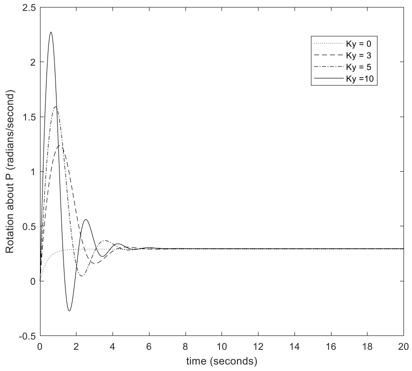
Ky=10:



Forward Velocity (v_R)

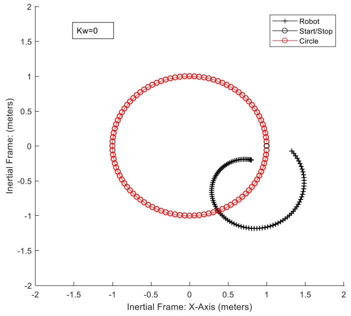


Rotational Velocity (ω_R)

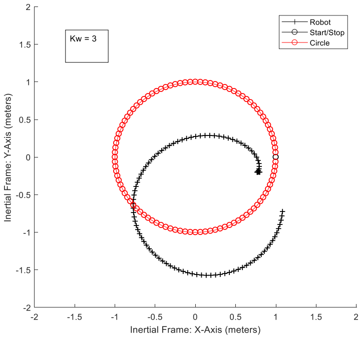


1-meter circle, 20 seconds, Kw Sensitivity, Kx=Ky=0, vg=0.3m/s, wg=0.3rad/s

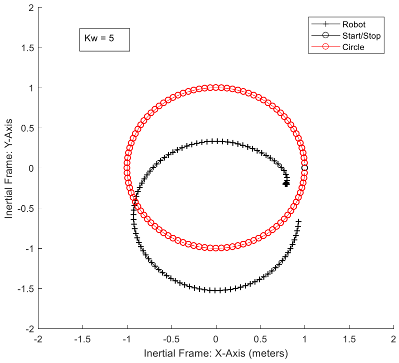
Kw=0:



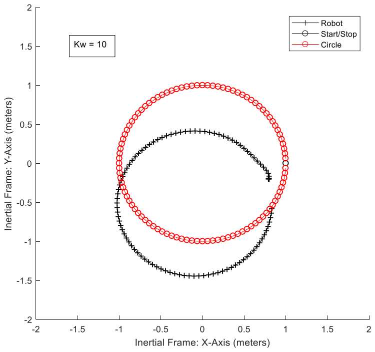
Kw=3:



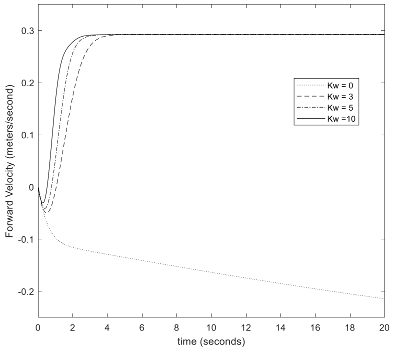
Kw=5:



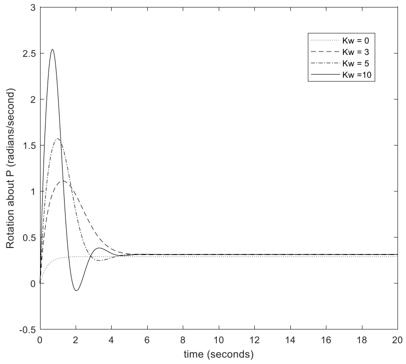
Kw=10:



Forward Velocity (v_R)

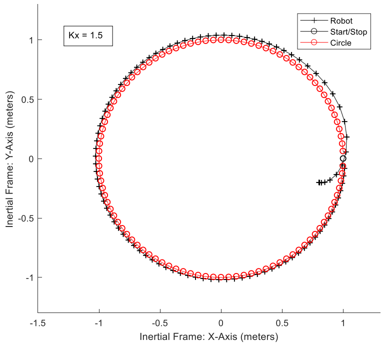


Rotational Velocity (ω_R)

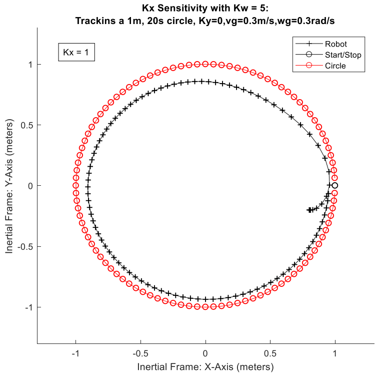


1 meter Circle Test: Kx Sensitivity with Kw = 5, Ky=0, Kx ranges 0.5→2, vg=0.3m/s and wg=0.3rad/sec:

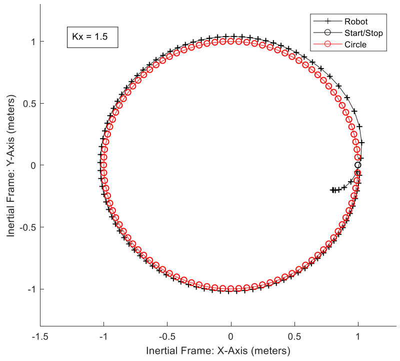
Kx=0.5:



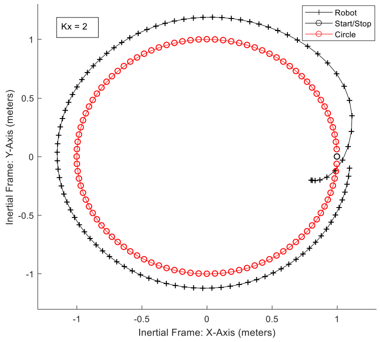
Kx=1:



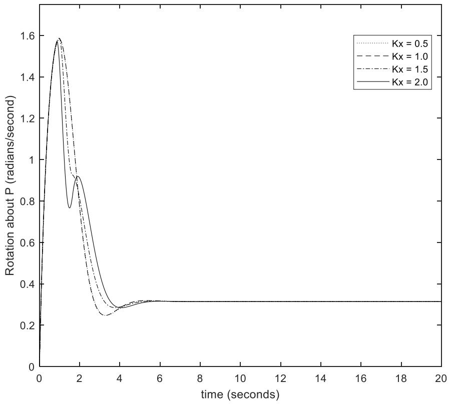
Kx=1.5:



Kx=2:

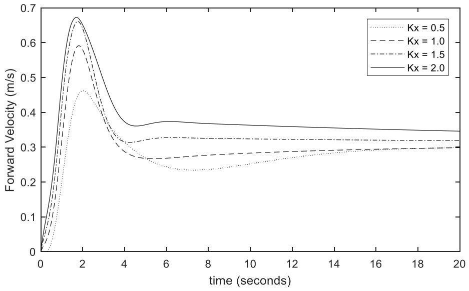


Rotational Velocity (ωr),for Kw=5,Ky=0, Kx .5→2



Forward Velocity (V_R), for Kw=5,Ky=0,

Kx .5→2

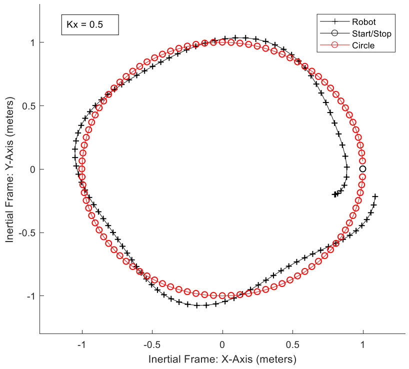


Tracking a 1 m, 20 s Circle

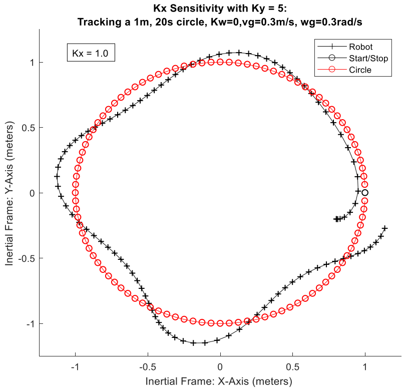
Using Kx and Kw

1 meter Circle Test: Kx Sensitivity with Ky = 5, Kw=0, Kx ranges 0.5→2, vg=0.3m/s and wg=0.3rad/sec:

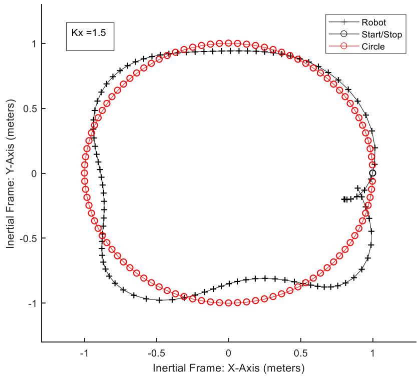
Kx=0.5:



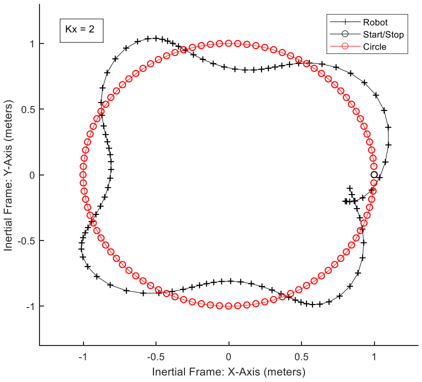
Kx=1:



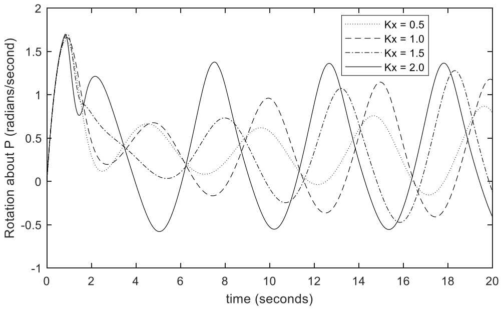
Kx=1.5:



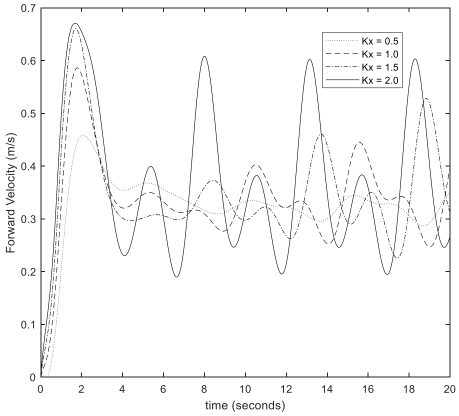
Kx=2:



Rotational Velocity (**ω**r),for Ky=4,Kw=0, Kx .5→2



Forward Velocity (Vxr), for Kw=5,Ky=0, Kx .5→2



Tracking a 1 m, 20 s Circle using

Kx and Ky

A3 – Arduino Code for Robotic Platform

The following code was written to apply the tracking control algorithm from the Kanayama paper. [6] It has the ability to track a 0.6 meter, 12 second circle and a 0.4 meter, 8 second circle.

First, an example of the output data delivered by the code after a typical run:

**************************ARDUINO OUTPUT *************************

```
0.6m
END
7132
5254
Gx,0.6,Gy,0.000,t,12.0
V,0.2630,W,0.8284
Ex,0.5021,
Ey,-0.0376,
Thta,7.7745
ex,0.0979,ey,0.0345,
exr,0.0422
240
0.004 , 0.000
0.028 , 0.000
0.060 , 1.016
0.096 , 1.012
0.136 , 1.008
0.180 , 1.016
0.220 , 0.008
0.256 , 0.036
0.288 , 0.072
0.308 , 0.112
0.320 , 0.156
0.324 , 0.196
0.320 , 0.236
0.312 , 0.272
0.296 , 0.300
0.284 , 0.324
0.264 , 0.348
0.248 , 0.368
0.232 , 0.384
0.212 , 0.400
0.196 , 0.412
0.176 , 0.424
0.156 , 0.436
0.136 , 0.448
0.116 , 0.456
0.096 , 0.464
0.072 , 0.472
0.052 , 0.476
0.028 , 0.480
0.008 , 0.484
1.012 , 0.488
0.988 , 0.488
0.964 , 0.488
0.940 , 0.484
0.916 , 0.480
0.892 , 0.476
0.868 , 0.472
0.844 , 0.464
0.820 , 0.456
0.796 , 0.448
0.776 , 0.436
0.752 , 0.424
0.732 , 0.408
```

0.712 , 0.396

0.692 , 0.380

0.676 , 0.360

0.656 , 0.344

0.640 , 0.324

0.624 , 0.304

0.612 , 0.284

0.596 , 0.260

0.584 , 0.240

0.572 , 0.216

0.564 , 0.192

0.556 , 0.168

0.548 , 0.144

0.540 , 0.116

0.536 , 0.092

0.532 , 0.064

0.528 , 0.036

0.528 , 0.012

0.528 , 1.012

0.528 , 0.984

0.532 , 0.960

0.536 , 0.932

0.540 , 0.908

0.548 , 0.880

0.556 , 0.856

0.564 , 0.832

0.576 , 0.808

0.588 , 0.784

0.600 , 0.764

0.616 , 0.740

0.632 , 0.720

0.648 , 0.700

0.664 , 0.680

0.684 , 0.660

0.704 , 0.644

0.724 , 0.628

0.744 , 0.612

0.768 , 0.596

0.788 , 0.584

0.812 , 0.572

0.836 , 0.564

0.860 , 0.552

0.888 , 0.544

0.912 , 0.540

0.936 , 0.532

0.964 , 0.528

0.988 , 0.524

1.016 , 0.524

0.012 , 0.524

0.040 , 0.524

0.064 , 0.528

0.092 , 0.532

0.116 , 0.540

0.140 , 0.548

0.168 , 0.556

0.192 , 0.564

0.216 , 0.576

0.240 , 0.588

0.264 , 0.600

0.284 , 0.612

0.308 , 0.628

0.328 , 0.644

0.344 , 0.664

0.364 , 0.680

0.380 , 0.700

0.400 , 0.724

0.412 , 0.744

0.428 , 0.764

0.440 , 0.788

0.452 , 0.812

0.464 , 0.836

0.472 , 0.860

0.480 , 0.884

0.488 , 0.912

0.492 , 0.936

0.496 , 0.960

0.500 , 0.988

***************************** **ARDUINO CODE** *******************************

```
/*
  Differential Drive Robotic Platform, with feedback control using shaft encoders
   No Array - Circle updated every timestep
   Start at [0;0;0]
   Case 1:
   0.6 m, 12 second circle, 10 ms per iteration loop
   Case 2:
   0.4 m, 8 second circle, 10 ms per iteration loop
   Encoders added to Pins 2,3 (interrupt operated)
   RMC enable motor 1 changed from Pin 3 to Pin 10
   -SRN-
*/

#include <EEPROM.h>

// Enable motor (PWM) outputs
#define EN1_PIN 10
#define EN2_PIN 11

// Motor direction outputs
#define DIR1_PIN 12
#define DIR2_PIN 13

//Encoder counters:
unsigned long Count1 = 0;
unsigned long Count2 = 0;

//Motor shaft velocity variables (rad/sec):
```

```
signed long phi1 = 0;

signed long phi2 = 0;

signed long phi1old = 0;

signed long phi2old = 0;

float phi1dot=0;//Wheel 1 velocity - rad/sec

float phi2dot=0;//Wheel 2 velocity


//Loop counters, address counter, real time variable, end time:

int  i=0;

byte k=10;

int addr=0;

float t=0;

int endtime=12; // seconds


//Approximate loop timestep and number of iterations during run:

float dt=0.01;   // (seconds)

int NUM=int(endtime/dt);   //(number of iteration steps in simulation)


//Robot Frame: Velocity Variables, Vel and Omega:

float Vel=0;  // (meter/sec)

float Omega=0;  //(rad/sec)


//Global Frame: Position and velocity variables:

float Ex=0.0;   //Start robot at 0 meters on X-axis

float Ey=0.0;  //Start robot at 0.0 meters on Y axis

float Theta=0.0;  //Start robot with angle orientation of 0 radians (0 degrees)

float Angref=1.571;  // Initial heading of circle is pi/2


//Motor, Battery, Controller and encoder constants:
```

```
const float Ka=0.0054;   //Armature constant for motors
const int GR=48;//Gear ratio
const float Gd=(float)1.0/dt;  // inverse of (dt) to avoid using arithmetic division inside of loop
const float Batt=0.164;  // Inverse of Battery voltage (1/6.1 volts)
const float R=7.874;  //inverse of twice the distance from wheel to center of bot (1/2*l in Siegwart), in meters
const float r1=0.03215;  //wheel radius in meters
int Vdc1=0;  //Motor 1 applied voltage
int Vdc2=0;  //Motor 2 applied voltage

//Controller Gains:
const int kx=1.1;
const int ky=6;
const int kw=8;
const float vref=0.2;
const float wref=0.2;

//Control error variables: (in meters)
float errx=0.0;   //global frame
float erry=0.0;
float etheta=0.0;
float errxr=0.0;   //robot frame
float erryr=0.0;


//Controlled entities (robot linear velocity and angular rotation velocity around platform center)
float v=0.0;  // meters/second
float w=0.0;  // radians/second
```

```
//Conversions from encoder pulses to radians:
const float radpuls=0.016362;

//A couple ending positions of the circle, in global coordinates
float Gx=0;
float Gy=0;

void setup()
{
  // Configure all motor outputs off for now
  pinMode(EN1_PIN, OUTPUT); digitalWrite(EN1_PIN, LOW);
  pinMode(EN2_PIN, OUTPUT); digitalWrite(EN2_PIN, LOW);
  pinMode(DIR1_PIN, OUTPUT); digitalWrite(DIR1_PIN, LOW);
  pinMode(DIR2_PIN, OUTPUT); digitalWrite(DIR2_PIN, LOW);
  pinMode(2, INPUT); // already have external pullup resistor
  pinMode(3, INPUT); // already have external pullup resistor

  // Both motors speed pins off for now
  analogWrite(EN1_PIN, 0);
  analogWrite(EN2_PIN, 0);

// Set an interrupt for each wheel encoder, "CHANGE" means 8 triggers per motor rev (0.0164 rad/trigger)
  attachInterrupt(digitalPinToInterrupt(2), encoder1, CHANGE);
  attachInterrupt(digitalPinToInterrupt(3), encoder2, CHANGE);

  // Variables for wheel velocities, clear during reset
    phi1=0; phi2=0;
  // Variables for desired motor voltages, clear during reset
```

```
    Vdc1=0;
    Vdc2=0;

  //Zero out Encoder counters if reset is pushed:
    Count1 = 0;
    Count2 = 0;
  // counters and timer restored if reset:
    i=0;
    t=0;

  Serial.begin(9600);
  delay(5000);
}

void loop()
{
  if (Serial.available() > 0) {
    int inByte = Serial.read();

    switch (inByte) {

    case '1': // Run 0.6m loop in 12 seconds
    NUM=1200;
    Serial.println("0.6m");
    //Enable motors with a small forward velocity:
    digitalWrite(DIR1_PIN, HIGH);  // Set Motor 1 forward direction
    digitalWrite(DIR2_PIN, HIGH);  // Set Motor 2 forward direction

    analogWrite(EN1_PIN, 50);    // Motor 1 on in forward direction
```

```
   analogWrite(EN2_PIN, 30);    // Motor 2 on in forward direction


  while (i<NUM) {
   while (k>0) {
   //Calculate wheel velocities, rad/sec, from change in shaft positions:
     phi1=Count1; //current shaft position, motor 1 (Count1 updated continuously via interrupt)
     phi2=Count2; //current shaft position, motor 2 (Count2 updated continuously via interrupt)
     phi1dot=(phi1-phi1old)*radpuls*Gd;//Delta phi/delta t, scaled by GR from motor shaft to
wheel
     phi2dot=(phi2-phi2old)*radpuls*Gd;
     phi1old=phi1;
     phi2old=phi2;


   //Calculate Robot Frame velocities in rad/sec, based on differential drive kinematics:
     Vel=0.5*r1*(phi1dot+phi2dot);
     Omega=R*r1*(phi1dot-phi2dot);  //R = 1/2*l


   //Calculate New Global Frame Postion:
     Theta=Theta+dt*Omega;
     Ex=Ex + dt*cos(Theta)*Vel;
     Ey=Ey + dt*sin(Theta)*Vel;


   //Calculate new position error in global frame, based on following a moving point on a circular
path:
     errx=0.6*cos(0.5236*t)-Ex;  //.6 meter radius, 12 second period, circular path
     erry=0.6*sin(0.5236*t)-Ey;
     etheta=Angref-Theta;


   //Calculate new position error in robot frame: e = inv(R(Theta))*[G-Ei]
```

```
    errxr=cos(Theta)*errx+sin(Theta)*erry;
    erryr=-sin(Theta)*errx+cos(Theta)*erry;

   // Enact Control Law for desired wheel velocities:
    v=vref*cos(etheta)+kx*errxr;
    w=wref+vref*(ky*erryr+kw*sin(etheta));

   // Calculate new motor voltage inputs from control velocities, and limit them to available battery voltage:
     Vdc1=int(255.0*GR*Ka*(30.3*v+1.92*w)*Batt); // Vdc ~ GR*Ka*phi = GR*Ka*inv(A)*[v;w]'
     Vdc2=int(255.0*GR*Ka*(30.3*v-1.92*w)*Batt);

     if (Vdc1>0) {
       digitalWrite(DIR1_PIN, HIGH);  // Set Motor 1 forward direction
     }
     if (Vdc1>255) {
       Vdc1=255;
     }
     if (Vdc1<0) {
       digitalWrite(DIR1_PIN, LOW);  // Set Motor 1 reverse direction
     }
     if (Vdc2>0) {
       digitalWrite(DIR2_PIN, HIGH);  // Set Motor 2 forward direction
     }
     if (Vdc2>255) {
      Vdc2=255;
     }
     if (Vdc2<0) {
       digitalWrite(DIR2_PIN, LOW);  // Set Motor 2 reverse direction
    }

 // Write the voltages to the motor controller card:
   analogWrite(EN1_PIN, Vdc1);    // Motor 1 on in forward direction
   analogWrite(EN2_PIN, Vdc2);    // Motor 2 on in forward direction

// Update real time variable and reference angle on circle
   t=t+dt;
   Angref=Angref+dt*0.5236;

  delay(9);// create a loop period of 10 ms, the minimum time to update encoder counts
  i++;  //Increment iteration counter
  k--;
 }
  k=10; // store X, Y location every o.1 seconds
  EEPROM.write(addr,int(Ex*250.0));
  EEPROM.write(addr+500,int(Ey*250.0));
  addr=addr+2;
}

  // Shut down and dump out some data about the run:
  analogWrite(EN1_PIN, 0);      // Motor 1 off
  analogWrite(EN2_PIN, 0);      // Motor 2 off
  Serial.println("END");
  Serial.println(Count1);
  Serial.println(Count2);
  Gx=0.6*cos(0.5236*t);
  Gy=0.6*sin(0.5236*t);
    Serial.print("Gx,");
    Serial.print(Gx,3);

    Serial.print(",Gy,");

    Serial.print(Gy,3);

    Serial.print(",t,");

    Serial.println(t,3);

    Serial.print("pdt1,");

    Serial.print(phi1dot);

    Serial.print(",pdt2,");

    Serial.println(phi2dot);

    Serial.print("V,");

    Serial.print(Vel,4);

    Serial.print(",W,");

    Serial.println(Omega,4);

    Serial.print("Ex,");

    Serial.print(Ex,4);

    Serial.print(",Ey,");

    Serial.print(Ey,4);

    Serial.print(",Thta,");

    Serial.println(Theta,4);

    Serial.print("ex,");

    Serial.print(errx,4);

    Serial.print(",ey,");

    Serial.print(erry,4);

    Serial.print(",exr,");

    Serial.println(errxr,4);

    Serial.println(addr);


   // Pull out and print the 0.1 second  X, Y location information

    for (int j=0; j<=addr;j=j+2) {
      Ex=EEPROM.read(j)/250.0;
      Ey=EEPROM.read(j+500)/250.0;
      Serial.print(Ex,3);Serial.print(" , ");Serial.println(Ey,3);
      }
      //Zero out the pertinent data for a new run

NUM=0;i=0;t=0;phi1=0;phi2=0;phi1old=0;phi2old=0;Count1=0;Count2=0;Ex=0.0;Ey=0.0;Theta=0
.0;addr=0;


  break;


  case '2': // 0.4m, 8 second circle
  NUM=800;
   Serial.println("0.4m");
   //Enable motors with a small forward velocity:
   digitalWrite(DIR1_PIN, HIGH);  // Set Motor 1 forward direction
   digitalWrite(DIR2_PIN, HIGH);  // Set Motor 2 forward direction

   analogWrite(EN1_PIN, 50);    // Motor 1 on in forward direction
   analogWrite(EN2_PIN, 30);    // Motor 2 on in forward direction

  while (i<NUM) {

   //Calculate wheel velocities, rad/sec, from change in shaft positions:
     phi1=Count1; //current shaft position, motor 1 (Count1 updated continuously via interrupt)
     phi2=Count2; //current shaft position, motor 2 (Count2 updated continuously via interrupt)
     phi1dot=(phi1-phi1old)*radpuls*Gd;//Delta phi/delta t, scaled by GR from motor shaft to
wheel
     phi2dot=(phi2-phi2old)*radpuls*Gd;
     phi1old=phi1;
```

```
    phi2old=phi2;

  //Calculate Robot Frame velocities in rad/sec, based on differential drive kinematics:
    Vel=0.5*r1*(phi1dot+phi2dot);
    Omega=R*r1*(phi1dot-phi2dot);  //R = 1/2*l

  //Calculate New Global Frame Postion:
    Theta=Theta+dt*Omega;
    Ex=Ex + dt*cos(Theta)*Vel;
    Ey=Ey + dt*sin(Theta)*Vel;

  //Calculate new position error in global frame, based on following a moving point on a circular path:
    errx=0.4*cos(0.7854*t)-Ex;  //.4 meter radius, 8 second period, circular path
    erry=0.4*sin(0.7854*t)-Ey;
    etheta=Angref-Theta;

  //Calculate new position error in robot frame: e = inv(R(Theta))*[G-Ei]
    errxr=cos(Theta)*errx+sin(Theta)*erry;
    erryr=-sin(Theta)*errx+cos(Theta)*erry;

  // Enact Control Law for desired wheel velocities:
    v=vref*cos(etheta)+kx*errxr;
    w=wref+vref*(ky*erryr+kw*sin(etheta));

  // Calculate new motor voltage inputs from control velocities, and limit them to available battery voltage:
     Vdc1=int(255.0*GR*Ka*(30.3*v+1.92*w)*Batt); // Vdc ~ GR*Ka*phi = GR*Ka*inv(A)*[v;w]'
     Vdc2=int(255.0*GR*Ka*(30.3*v-1.92*w)*Batt);
     if (Vdc1>0) {
        digitalWrite(DIR1_PIN, HIGH);  // Set Motor 1 forward direction
     }
     if (Vdc1>255) {
        Vdc1=255;
     }
     if (Vdc1<0) {
        digitalWrite(DIR1_PIN, LOW);  // Set Motor 1 reverse direction
     }
     if (Vdc2>0) {
        digitalWrite(DIR2_PIN, HIGH);  // Set Motor 2 forward direction
     }
     if (Vdc2>255) {
       Vdc2=255;
     }
     if (Vdc2<0) {
        digitalWrite(DIR2_PIN, LOW);  // Set Motor 2 reverse direction
     }

  // Write the voltages to the motor controller card:
    analogWrite(EN1_PIN, Vdc1);    // Motor 1 on in forward direction
    analogWrite(EN2_PIN, Vdc2);    // Motor 2 on in forward direction


 // Update real time variable and reference angle on circle
    t=t+dt;
    Angref=Angref+dt*0.7854;


   delay(9);// create a loop period of 10 ms, the minimum time to update encoder counts
```

```
   i++;  //Increment iteration counter
 }


   // Shut down and dump out some data about the run:
   analogWrite(EN1_PIN, 0);      // Motor 1 off
   analogWrite(EN2_PIN, 0);      // Motor 2 off
   Serial.println("END");
   Serial.println(Count1);
   Serial.println(Count2);
   Gx=0.4*cos(0.7854*t);
   Gy=0.4*sin(0.7854*t);
    Serial.print("Gx,");
    Serial.print(Gx,3);
    Serial.print(",Gy,");
    Serial.print(Gy,3);
    Serial.print(",t,");
    Serial.println(t,3);
    Serial.print("pdt1,");
    Serial.print(phi1dot);
    Serial.print(",pdt2,");
    Serial.println(phi2dot);
    Serial.print("V,");
    Serial.print(Vel,4);
    Serial.print(",W,");
    Serial.println(Omega,4);
    Serial.print("Ex,");
    Serial.print(Ex,4);
    Serial.print(",Ey,");
    Serial.print(Ey,4);
```

```
      Serial.print(",Thta,");
      Serial.println(Theta,4);
      Serial.print("ex,");
      Serial.print(errx,4);
      Serial.print(",ey,");
      Serial.print(erry,4);
      Serial.print(",exr,");
      Serial.println(errxr,4);

      //Zero out pertinent data for the next run
     NUM=0;i=0; t=0;phi1=0; phi2=0;phi1old=0; phi2old=0;Count1 = 0;Count2 = 0;Ex=0.0;Ey=0.0;Theta=0.0;

    break;

    default:
    Serial.println("X");
  }
 }
}

void encoder1() {
  Count1++;
}
void encoder2() {
  Count2++;
}
```