Carnegie Mellon University

Team 5
Final Presentation

Aditya, Lakshmi, Nabarun, Rachit, Sumedh

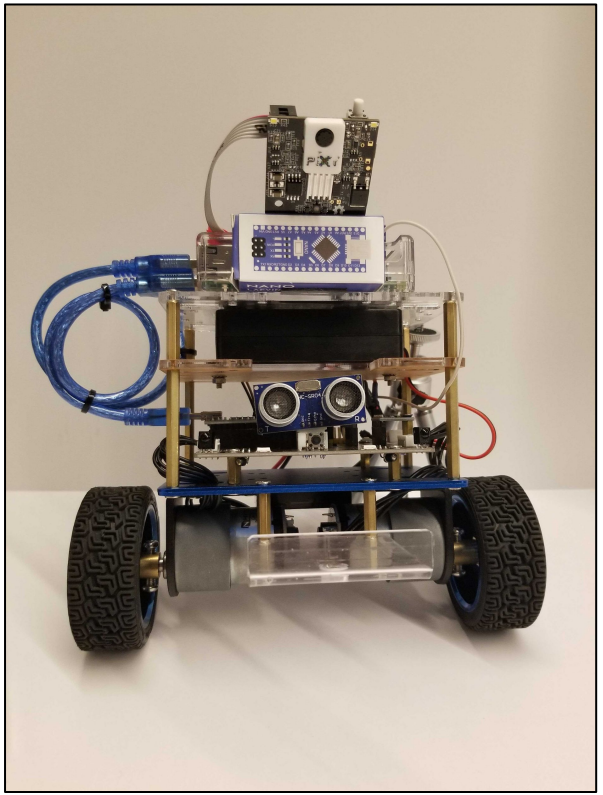
Problem Definition - Tumbller AGV

Problem Definition

- Automated guided vehicles (AGVs) have been used in factories and other industrial settings since 1953[1]
- They rely on environmental markers and follow a predetermined route
- The biggest challenges plaguing their deployment are payload and navigation

Project Objectives

- Self balancing MPC robot
- Inner loop PD balance controller
- Obstacle detection and avoidance
- Follower robot



Carnegie Mellon University

Literature Review

Title of Paper	Author	Content
Model predictive control of systems with deadzone and saturation	Giacomo Galuppini, Lalo Magni, Davide Martino Raimondo	Authors implement and test two different control strategies on MPC. The former relies on hybrid MPC, later is based on deadzone inversion and standard MPC.
Model predictive control of a Differential-Drive Mobile Robot	Samir Bouzoualegh, El-Hadi Guechi and Ridha Kelaiaia	Linearised MPC was implemented for differential drive robot based on dynamic model. Predictive control law gains were acquired by minimizing a quadratic criterion.
MPC for path Following Problems of Wheeled Mobile Robots	Shuyou Yu, Yang Guo, Lingyu Meng, Ting Qu, Hong Chen	Observer based model predictive control for path following problems of wheeled mobile robots with input disturbances is proposed.
Model Predictive Control of a Wheeled Inverted Pendulum Robot	Navid Dini, Vahid Johari Majd	Generalised predictive control (CARIMA model) has been applied to a segway like robot and compared wit classical controls
Joe: a mobile, inverted pendulum	F. Grasser, A. D'arrigo, S. Colombi, and A. C. Rufer	Simplified the control system design of JOE robot by decoupling vehicle dynamics into two subsystems (yaw and pitch), which permitted the design of a state feedback algorithm via pole placement for each subsystem.
Low cost two-wheels self-balancing robot for control education	C. Gonzalez, I. Alvarado, and D. M. La Peña	Combined two linear controllers; LQR for angular speed and tilt while PI for angular acceleration.LQR controller is used to control angular speed and tilt angle of the wheel. Then, angular acceleration of the wheel was referred to PI controller for magnitude control. There are two PI controller, one for each wheel
Stabilizing Two-Wheeled Robot Using Linear Quadratic Regulator and States Estimation	Nur Uddin, Teguh Aryo Nugroho, and Wahyu Agung Pramudito	

System Modelling

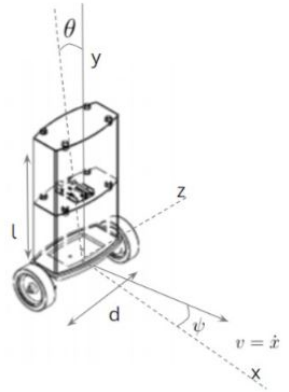


Fig. 1. Model of wheeled inverted pendulum robot

$$F_l + F_r = (m_{cart} + m_{pend})\ddot{x} + f\dot{x} + m_{pend}\ddot{\theta}cos\theta - m_{pend}l\dot{\theta}^2 sin\theta \quad (1)$$

$$(I_{pend} + m_{pend}l^2)\ddot{\theta} + m_{pend}glsin\theta = -m_{pend}l\ddot{x}cos\theta \quad (2)$$

$$(I_{pendY} + (\frac{I_w}{r^2} + m_{wheel}).d^2)\ddot{\psi} = d(F_l - F_r) \quad (3)$$

where

$$F_i = \frac{\tau_i}{r} = \frac{k_T V_i}{Rr}$$

Velocity and Tilt model

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & t_1 & t_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & t_3 & t_4 & 0 \end{bmatrix} B_1 = \begin{bmatrix} 0 \\ t_5 \\ 0 \\ t_6 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} D_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$States = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T$$

Heading Angle model

$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} B_2 = \begin{bmatrix} 0 \\ t_7 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} D_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$States = \begin{bmatrix} \psi & \dot{\psi} \end{bmatrix}^T$$

where,

$$t_1 = \frac{-(I_{pend} + m_{pend}l^2)f}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_2 = \frac{m_{pend}^2 gl^2}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_3 = \frac{-m_{pend}lf}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_4 = \frac{m_{pend}gl(m_{cart} + m_{pend})}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_5 = \frac{I_{pend} + m_{pend}l^2}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_6 = \frac{m_{pend}l}{I_{pend}(m_{cart} + m_{pend}) + m_{cart}m_{pend}l^2}$$

$$t_7 = \frac{d}{(I_pend + d^2.(m_{wheel} + \frac{I_w}{r^2})}$$

Carnegie Mellon University

MPC Formulation - Feedback Loop

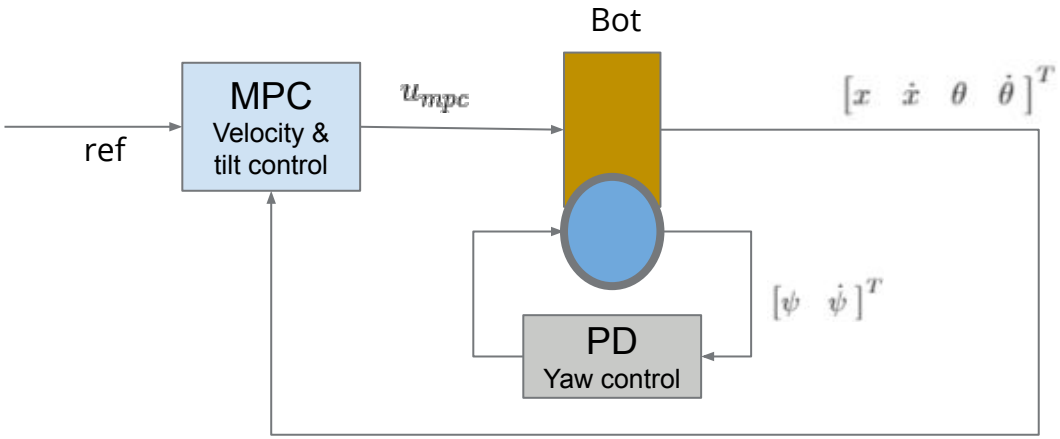
- MPC designed to provide the optimal control input to control the velocity/position of the robot in the loop.

Constraints:

$0.15\ \ rads < \theta < 0.15\ \ rads$

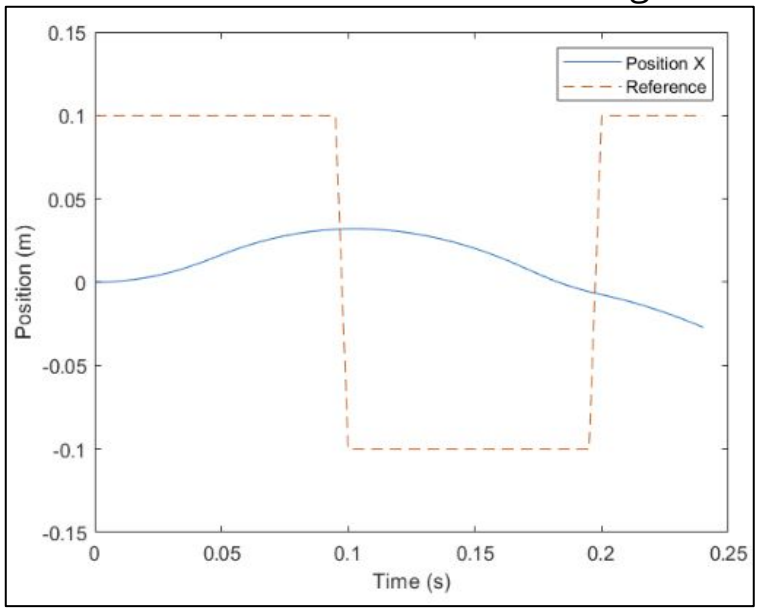
$-12\ V < u < 12\ V$

N = 10



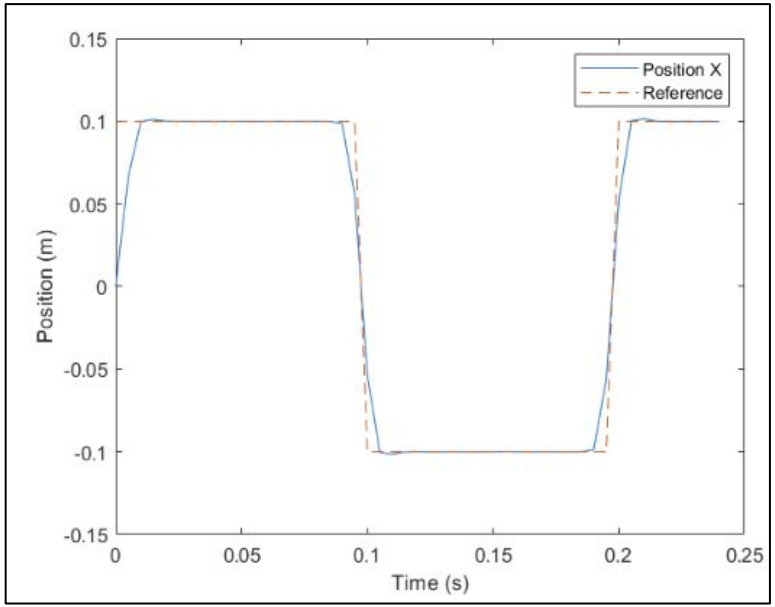
Square Wave Tracking (Linearized model)

Position Reference Tracking



$$r(i) = X_{ref}(i)$$

Velocity Reference Tracking



$$r(i) = \frac{X_{ref}(i+1) - X_{ref}(i)}{dt}$$

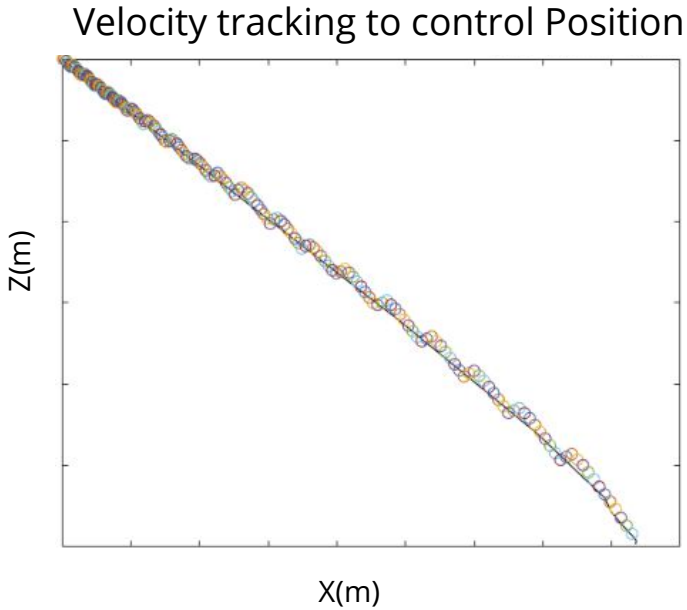
States: $[x \;\; \dot{x} \;\; \theta \;\; \dot{\theta}]$ $[\dot{x} \;\; \theta \;\; \dot{\theta}]$ $x = x + \dot{x}.dt$

Carnegie Mellon University

Full-bot tracking (Linearized model)

- MPC for velocity tracking
- Proportional control for heading angle
- Hard Constraints on limits of Tilt and Control Effort

- Unable to turn through sharp corners
- Eventually the tilt goes beyond recoverable ranges
  - Both in case of soft and hard constraints



MPC Formulation - Feedforward reference

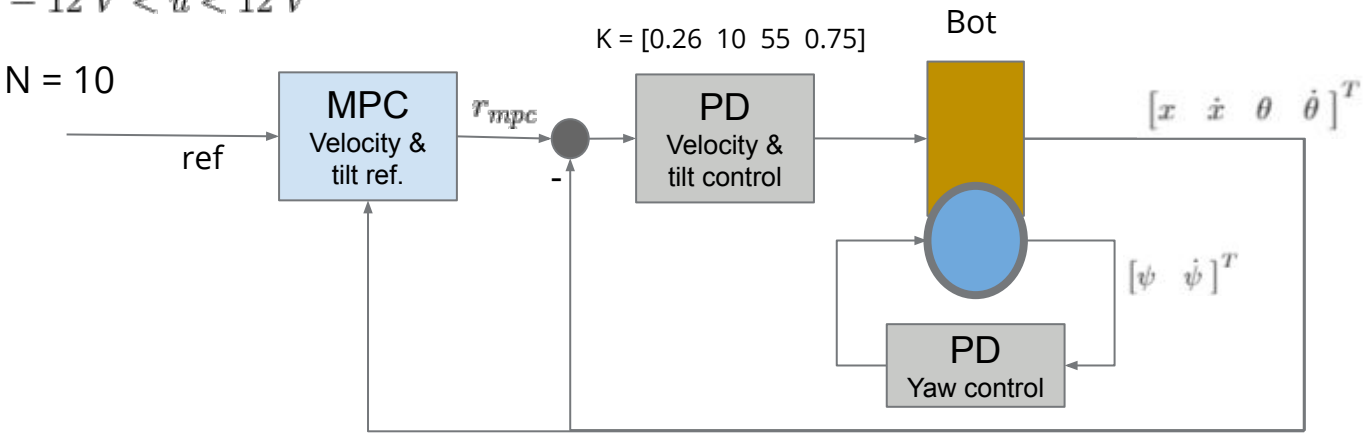
MPC designed to provide the optimal reference to be tracked by the inner PD loop.

MPC responsible for only position/reference tracking.

Constraints:

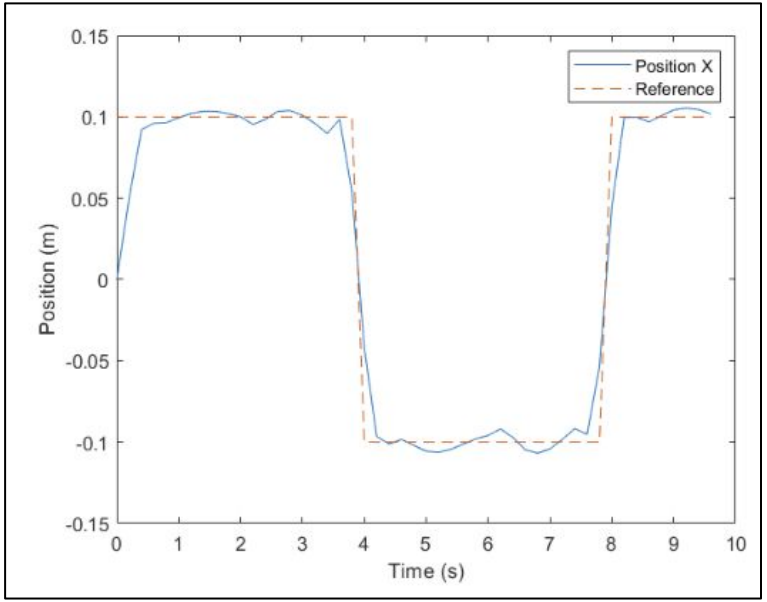
$0.15\ \ rads < \theta < 0.15\ \ rads$

$-12\ V < u < 12\ V$

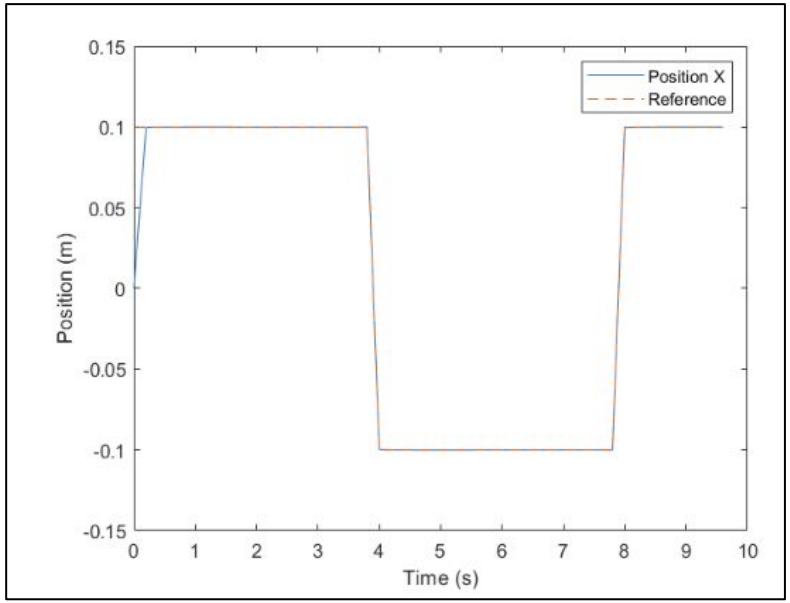


Square Wave Tracking (Linearized model)

Position Reference Tracking



Velocity Reference Tracking



$$r(i) = X_{ref}(i)$$

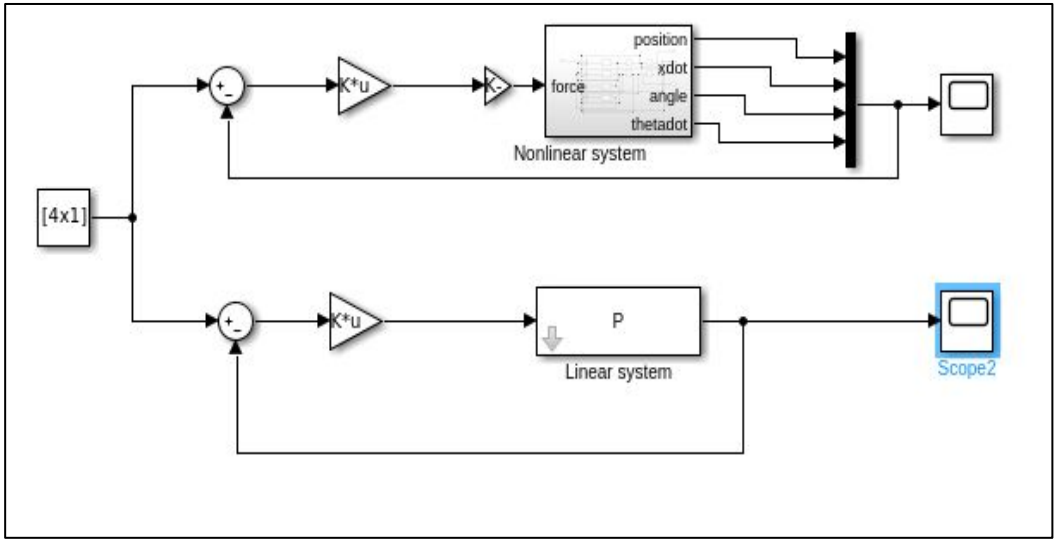
$$r(i) = \frac{X_{ref}(i+1) - X_{ref}(i)}{dt}$$

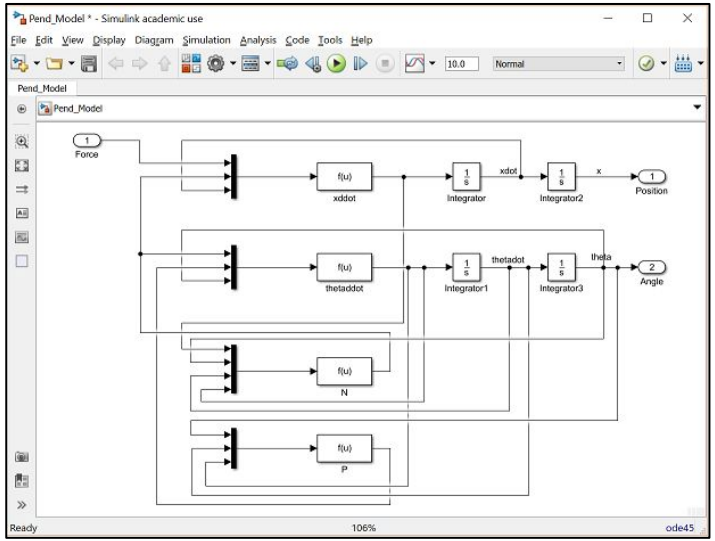
States: $[x \;\; \dot{x} \;\; \theta \;\; \dot{\theta}]$

$[\dot{x} \;\; \theta \;\; \dot{\theta}]$ $\qquad x = x + \dot{x}.dt$

Carnegie Mellon University

Simulation

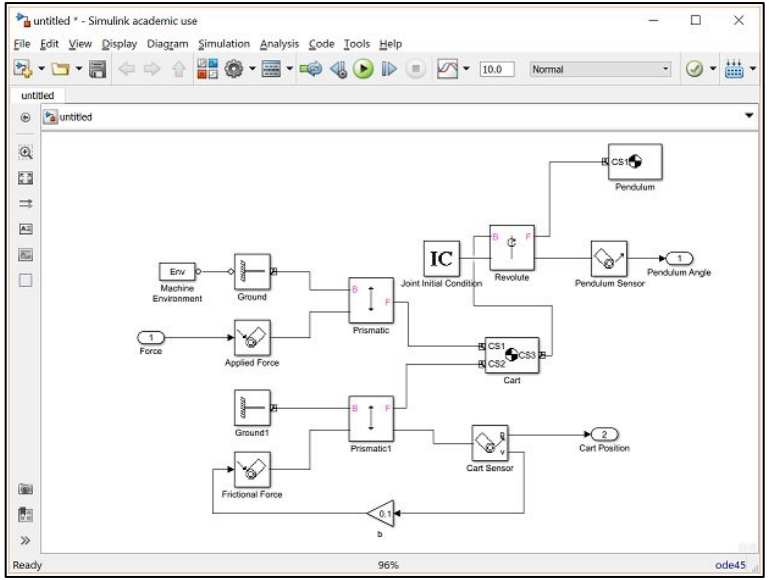




Carnegie Mellon University

Simulation - Simscape

- Non-linear Simscape model
- MPC as controller for system with [velocity, angle, angular velocity] as state space.
- MPC as controller for system with [position, velocity, angle, angular velocity] as state space.
- MPC (both the above state space) as reference tracking with inner PD controller

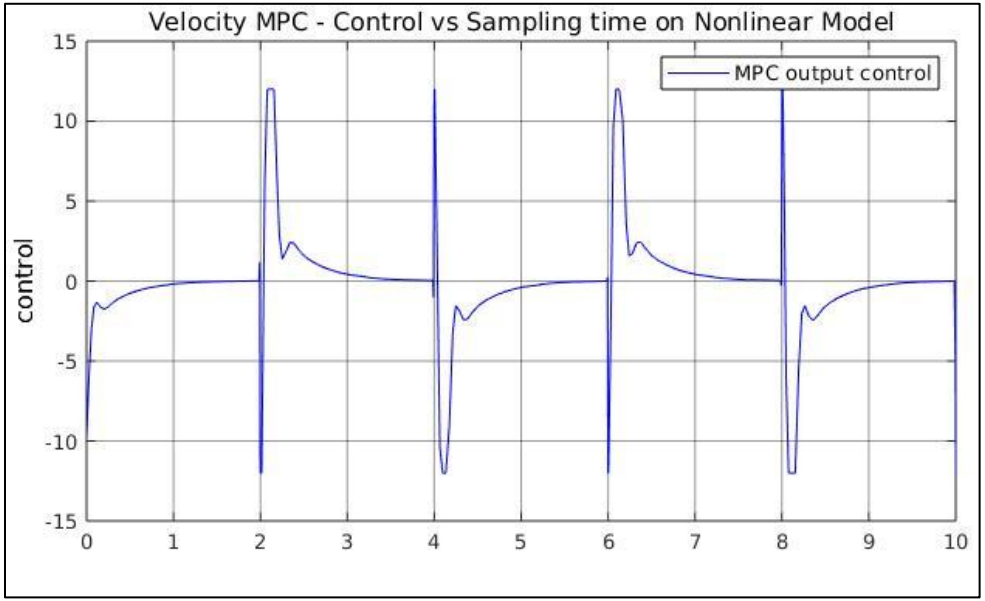


Results of Simulation

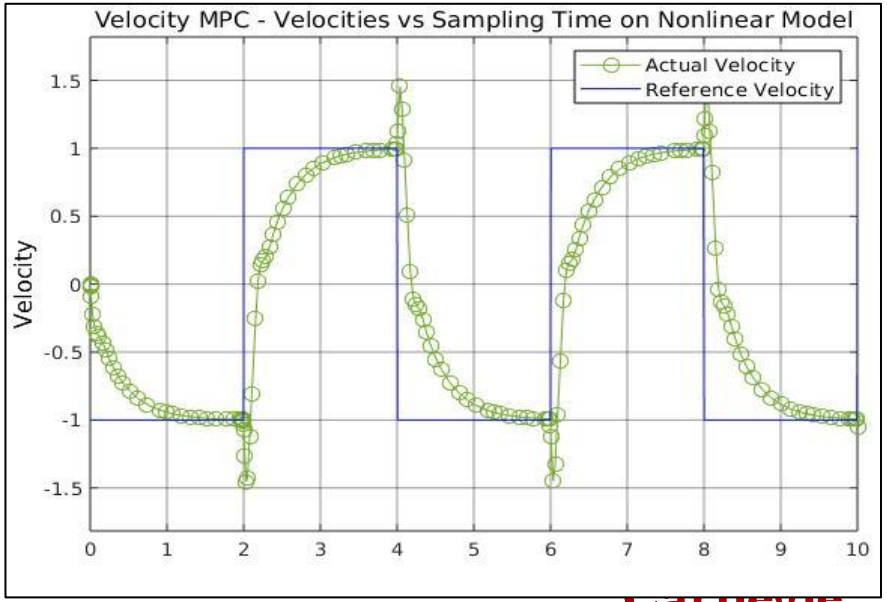
https://drive.google.com/file/d/1rjMO3LJKOsm5m7tlLZGzf2hqRbgwGdR2/view?usp=sharing

Results of Simscape

Control

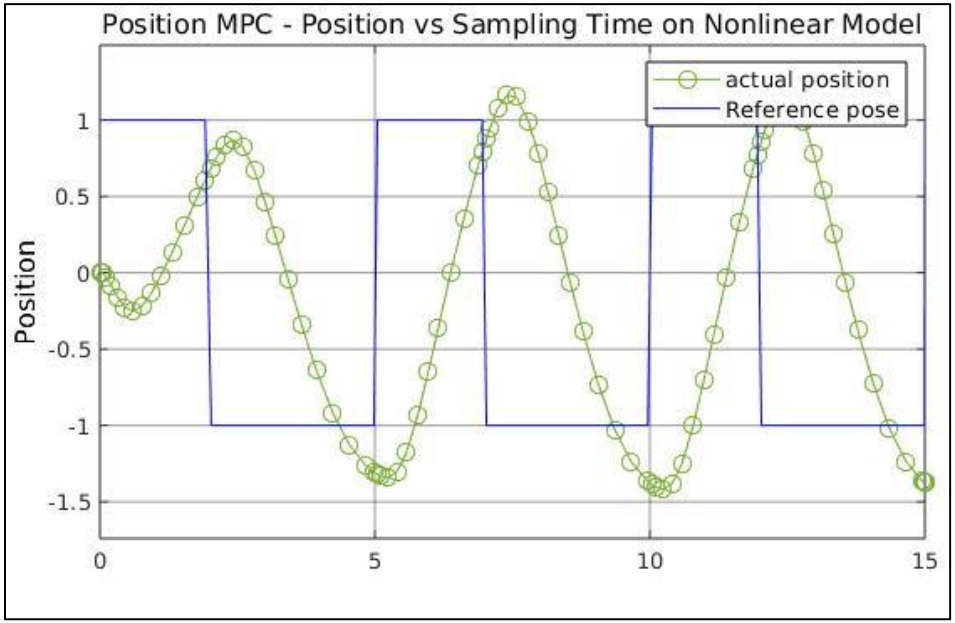


Velocity - Reference vs System Output

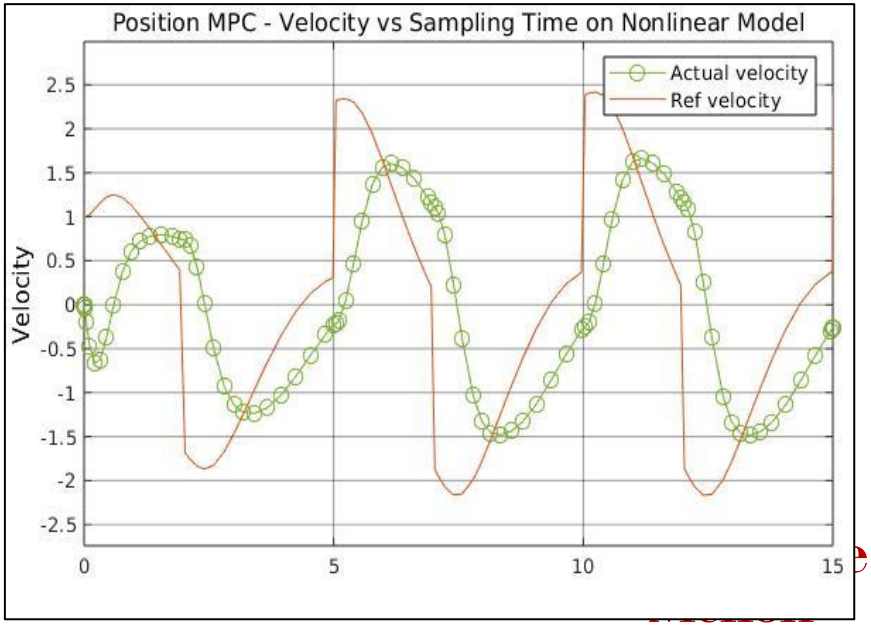


Results of Simscape

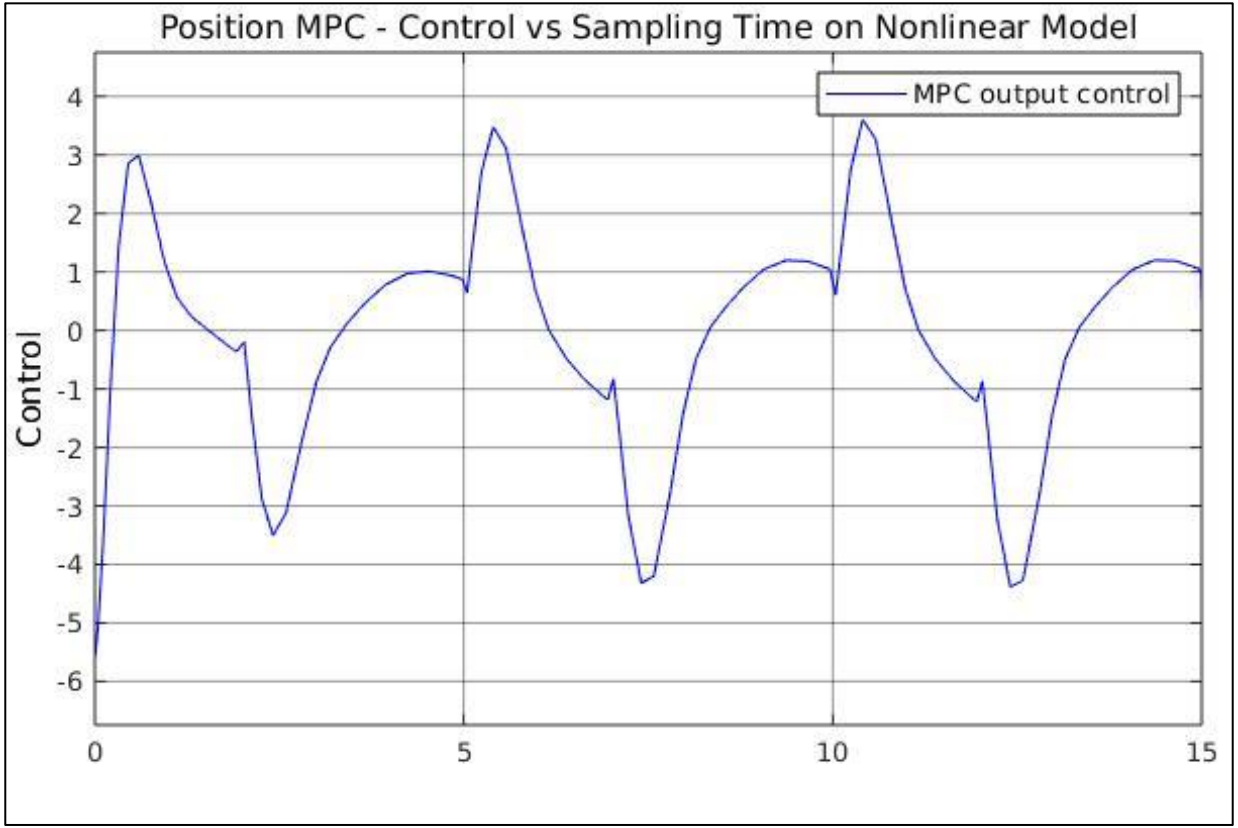
Position - Reference vs System Output



Velocity - Reference vs System Output



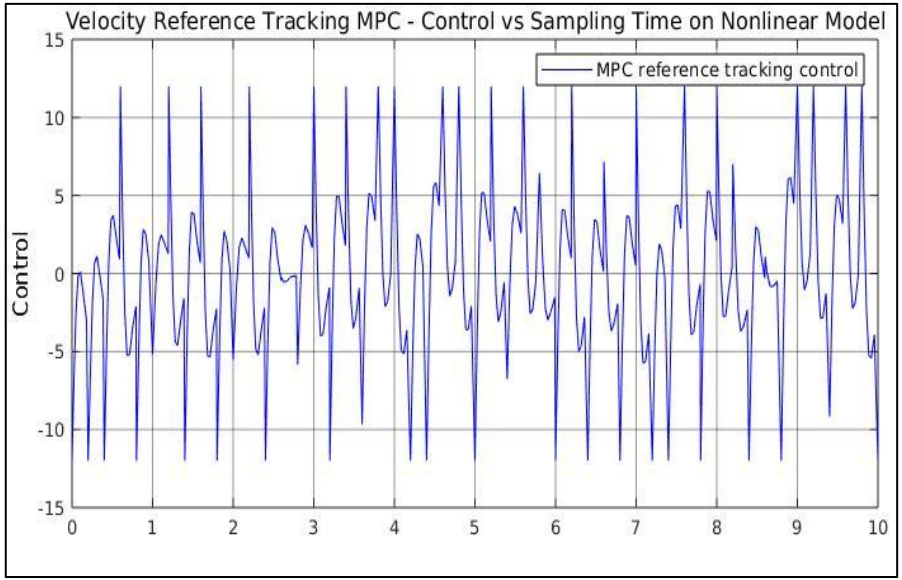
Results of Simscape



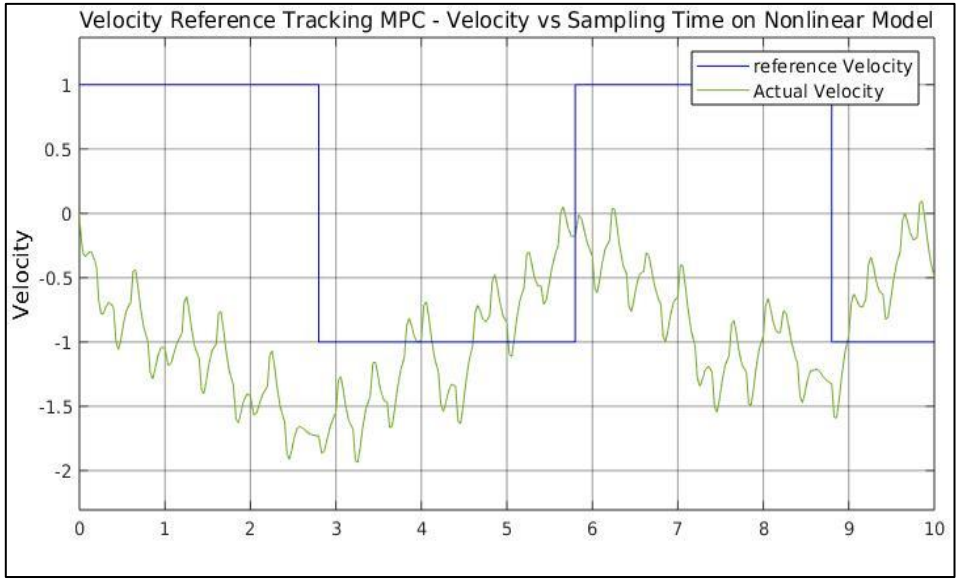
Carnegie Mellon University

Results of Simulation

Control



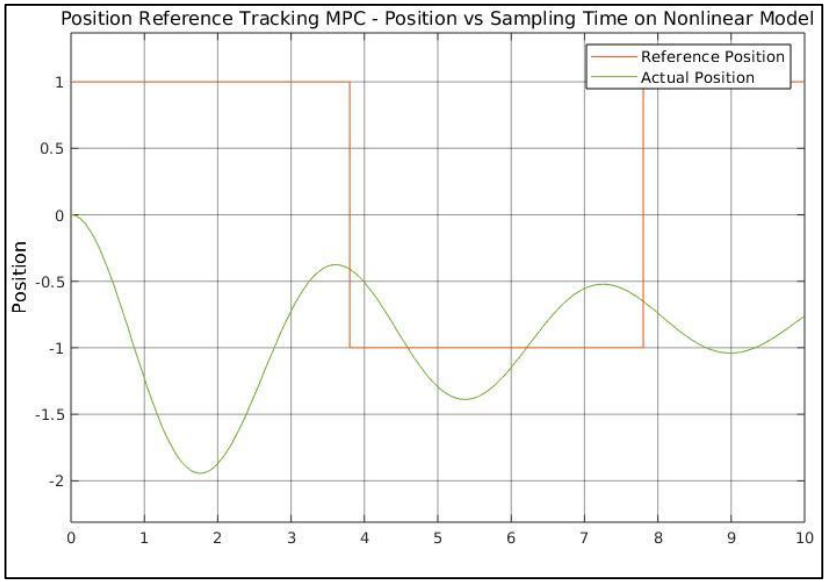
Velocity - Reference vs System Output



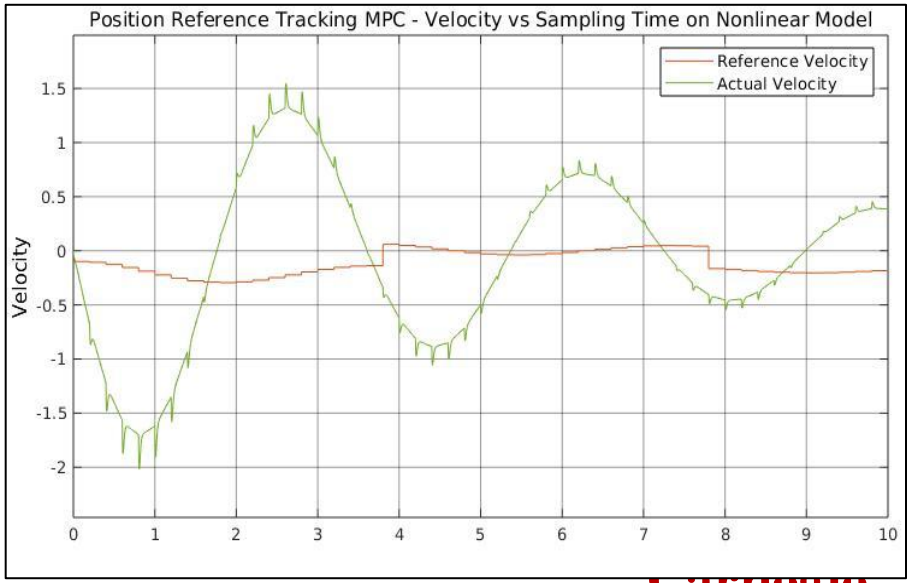
Carnegie Mellon University

Results of simulation

Position - Reference vs System Output



Velocity - Reference vs System Output



Carnegie Mellon University

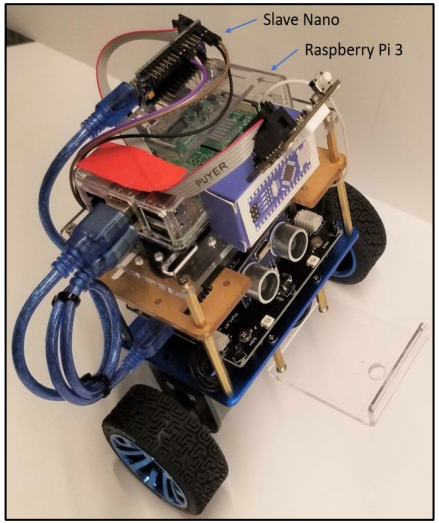
Approach

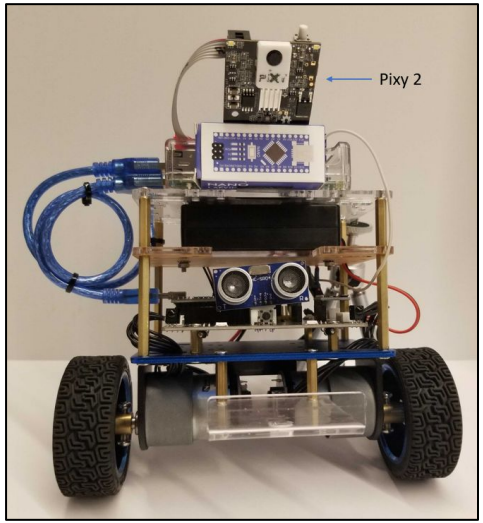
Self-balancing full robot MPC with obstacle avoidance:

- Linear dynamic model with linear MPC and inner PD balance and tracking controller: Hardware implementation
- Linear dynamic model with full state MPC with an inner PD controller: Simulation
- Non-Linear dynamic model with full state MPC with an inner PD controller: Simulation

Follower robot:

- Ultrasonic based distance tracking
- Master robot followed by single/multiple robots
- Possibility to extend MPC control to follower robots through Bluetooth/Raspberry Pi





Carnegie Mellon University

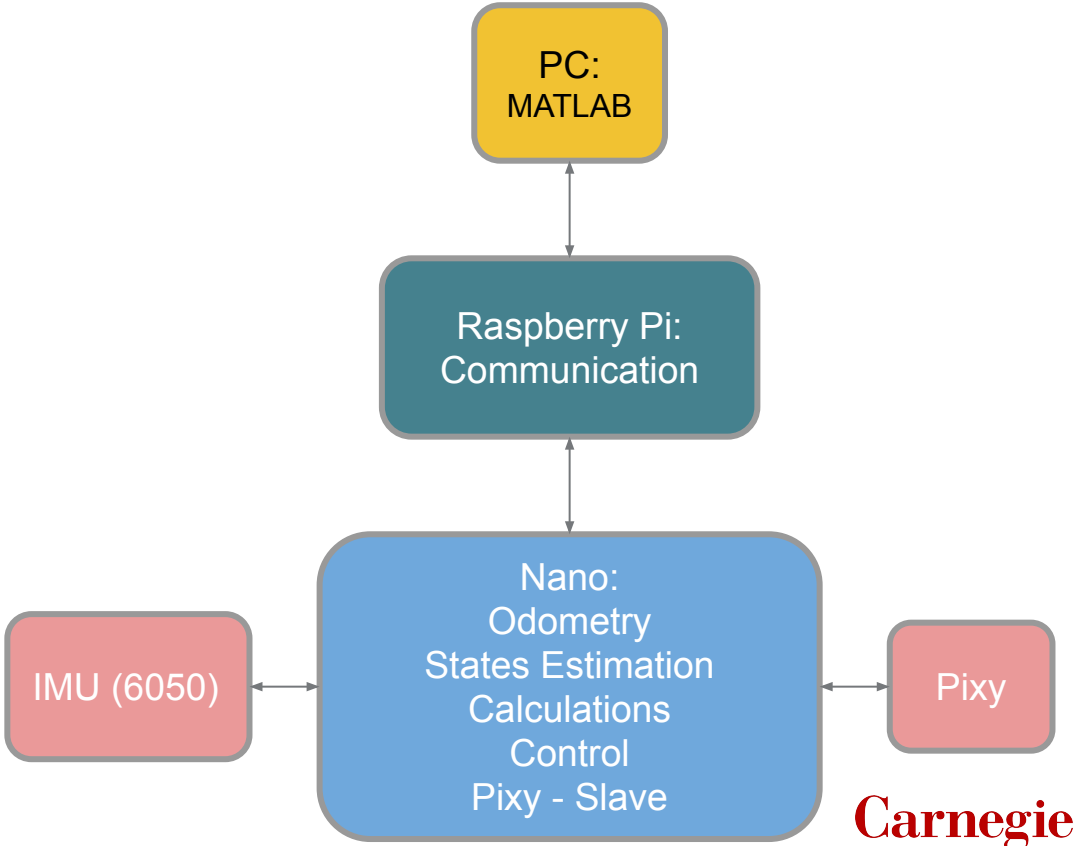
Hardware

Sensors:

- PixyCam 2
- Ultrasonic sensor for obstacle detection
- Wheel encoders for odometry
- IMU for orientation
- Raspberry Pi for communication

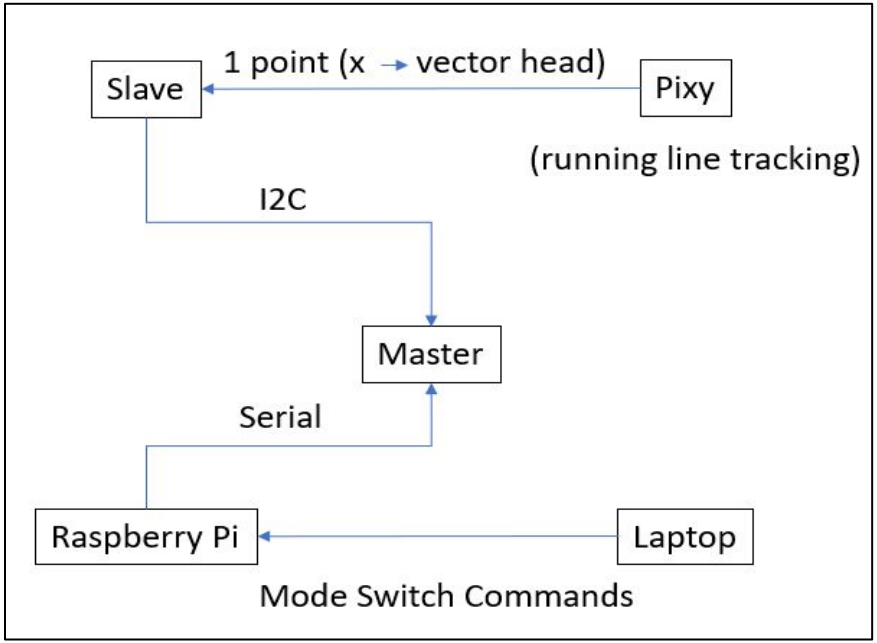
Controllers:

- Arduino Nano - Master: main control loop
- Arduino Nano - Slave: PixyCam2 data processing
- Laptop - MPC loop

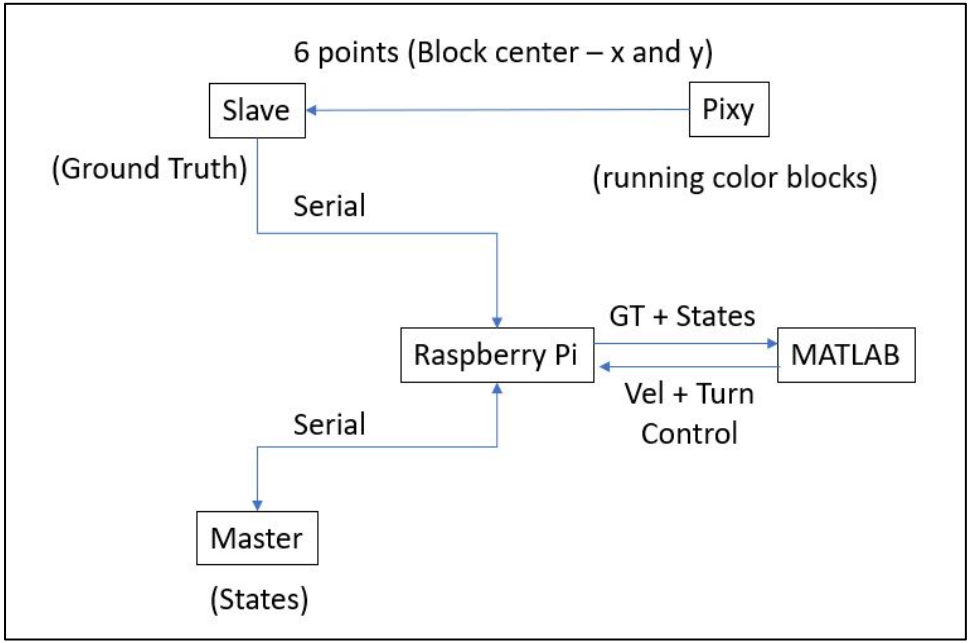


Carnegie Mellon University

Communication



Line Tracking



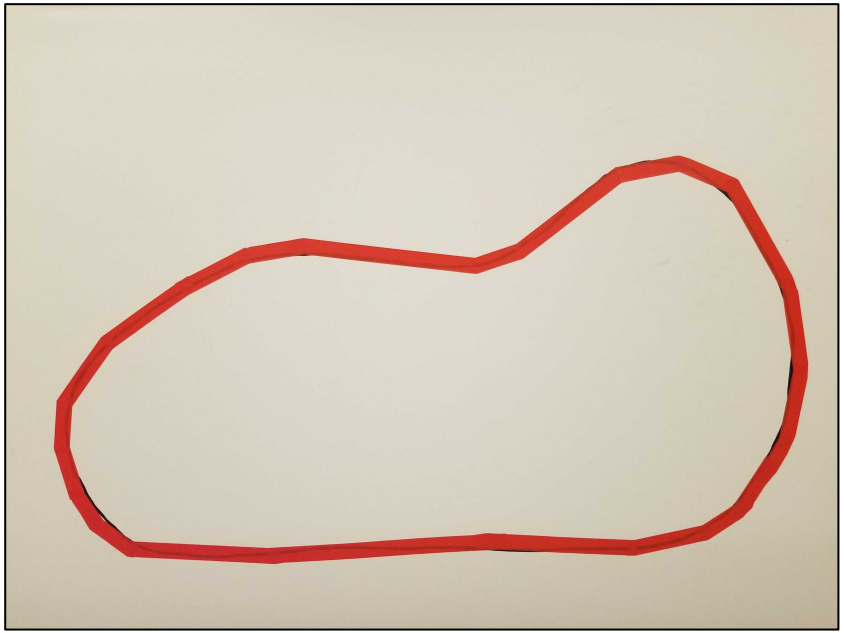
MPC Line Tracking

Carnegie Mellon University

Demo - Test Track - PD Controller

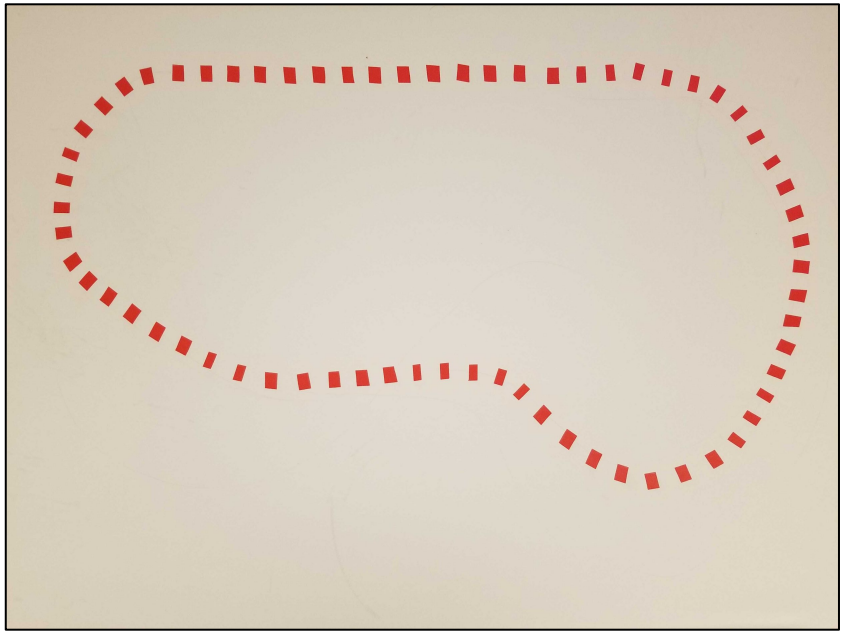
Demo Tracks:

- Continuous track for line following
- Discrete track for MPC



Carnegie Mellon University

Demo - Test Track - MPC



Demo - PD Line Tracking + Obstacle Detection

https://youtu.be/Dc_DUGhrgQ4

Demo - PD Line Tracking + Follower Robot

https://youtu.be/XDkgczZaCCk

Linear MPC Reference Tracking

https://youtu.be/pvhCd1TMDjY

Analysis

- Simulation and hardware match for MPC feedforward as seen in our hardware testing.
- In the simulation and hardware the reference is being tracked with velocity, tilt and rate of tilt as state space.
- We used 2 different models for MPC simulation. It was easier to tune Q and R for MPC with less state space variables and then using that as a reference we tuned the position reference MPC for both feedforward and feedback MPC.
- Robot can successfully track a solid line with slight errors in yaw tracking.
- Line tracking with MPC failed due to delay of >200ms required in processing the data. During this time Pixy loses track of the line.

Challenges

Communication:

- Parsing strings in Master (Arduino Nano) for MPC
- Inherent delay when processing strings in MATLAB

Systems:

- Syncing PixyCam, odometry and IMU data
- Communication and timing challenge between MATLAB and Raspberry Pi

Mechanics:

- System torque output is limited
- Wheel slip during balancing and tracking
- Wheels don’t spin at the same speed

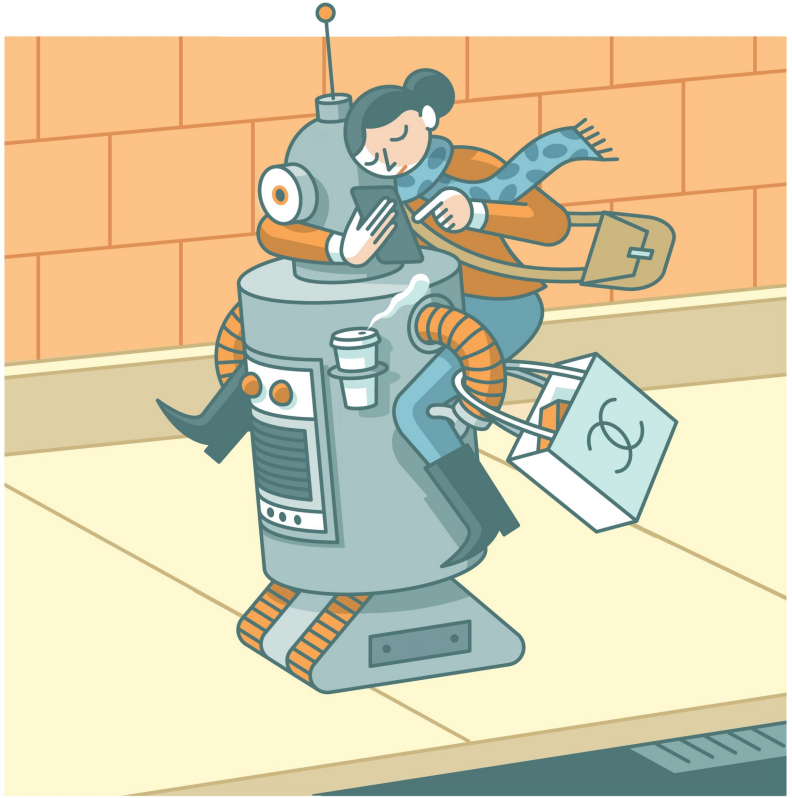
Carnegie Mellon University

Thank you!
Question?

Link to GitHub

https://github.com/lmaddira/Adv-controls

Carnegie Mellon University



Carnegie Mellon University