# Self-Balancing Robot Model Predictive Control

Aditya Patra
*dept. Mechanical Engineering*
Pittsburgh
adityapa@andrew.cmu.edu

Lakshmi Pranathi Maddirala
*dept. Mechanical Engineering*
Pittsburgh
lmaddira@andrew.cmu.edu

Nabarun Banerjee
*dept. Mechanical Engineering*
Pittsburgh
nabarunb@andrew.cmu.edu

Rachit Mishra
*dept. Mechanical Engineering*
Pittsburgh
rachitm@andrew.cmu.edu

Sumedh Vaidya
*dept. Mechanical Engineering*
Pittsburgh
smvaidya@andrew.cmu.edu

***Abstract*—This paper presents the work done on designing and implementing a nonlinear model predictive control (MPC) based controller on a self-balancing two wheeled mobile robot for a dynamic environment. The inverted pendulum model used in this study is naturally an unstable system when subjected to natural disturbances due to environmental loading conditions. This model has been studied extensively in the past for PD controllers but MPC has been relatively left unexplored. In this paper we propose various controllers, PD controller, MPC controller as feedback and MPC reference tracking feed forward controller with a inner PD controller, to drive the balancing two wheeled robot in a dynamic environment. Simulation results demonstrate the feasibility and efficiency of our proposed design and experimental results on the hardware validate the these results. A follower robot and obstacle detection was added in the final demonstration to show multi robotics interaction.The PD controller with live camera feedback was able to demonstatet robust line tracking, obstacle detection and multi-robot interaction.The MPC based self balancing robot was able to track a straight line with full state feedback.**

**Keywords: Wheeled inverted pendulum; Segway Robot; Model predictive control, Self-balancing robot;**

## I. Introduction

Two wheeled balancing robot have become very popular since the introduction of the Segway. In terms of mechanism design, most of these robots take inspiration from the inverted cart and pendulum system. The inverted pendulum is one of the most common problems encountered in the field of robotics and control systems. Implementing control algorithms for two wheeled self balancing robots is often deceptively simple as they need to move and balance at the same time. This makes these robots an interesting research topic for someone looking to enhance their classical control and feedback control skills. Generally, these robots employ a closed loop feedback control strategy which ensures that any errors in motion or tilt are quickly compensated for by the controller.

The absence of a third and/or a fourth wheel results in two wheeled robots having highly complicated dynamics when compared to three and four wheeled robots. However, these robots have certain advantages when compared to others such as their ability to manoeuvre in closed spaces. This is also helped by the fact that these robots have a smaller cross sectional area as compared to other robots. This makes them a natural pick for use in offices, houses and in elevators. As a result of their versatility, their applications include but are not limited to service industry, cleaning, warehousing and product handling.

## II. Problem Definition

Automated Guided Vehicles (AGVs) have been used in factories and other industrial settings since 1953. With an increase in their adoption and deployment multi-agent AGVs are on the rise. The biggest challenges plaguing their deployment are payload and navigation. They rely on environmental markers to follow a predetermined route. The rationale behind this project is to tackle the marker based navigation challenge while addressing multi-agent interactions such as collision and obstacle detection and avoidance along with dynamically changing operating environments. How accurately a robot tracks the given path is critical to the performance in an environment. The biggest applications of such a controller is in warehouse environments where they are expected to carry heavy loads and avoid crashing into one another. The team set out to tackle this problem by developing the following objectives: integrated outer loop full robot MPC control with reference tracking in the Elegoo Tumbller hardware with an inner loop PD controller for balancing. The final version of the robot would also need to have obstacle detection and multi-agent interactions such as following a master robot.

## III. Literature review

Model predictive control (MPC) originated in the late 70s and has developed considerably since then. MPC does not refer to a specific control strategy, but rather to vast range of control methods that explicitly use a model of the process in order to obtain the control signal by minimizing an objective function. These methods create controllers that have similar structures and retain the required degrees of freedom. MPC control has had a wide range of applications in the literature [7] [8] [9].

Different MPC algorithms only differ in the model used to represent the process, the noise and cost function to be minimized. Most systems have multiple variables that need to be controlled, these systems can be controlled using an MPC method called multi-variable generalized predictive control (GPC) [6].

Self-balancing robots have also been used as an interesting pedagogical teaching tool to enhance the learning of feedback control systems by implementing classical Proportional-Integral-Derivative (PID) and phase lead-lag controllers [13]. The authors in [14] simplified the control system design of the JOE robot by decoupling the vehicle dynamics into two subsystems (yaw and pitch), which permitted the design of a state feedback algorithm via pole placement for each of the subsystems.

Ivoilov, Zhamud Trubin [15] designed a complementary filter to estimate tilt angle estimation to overcome mid and low-frequency accelerometer distortions. Proportional–Integral–Derivative (PID) controller is used to balance tilt angle and preventing unpredictable movement of the system in the horizontal plane. Jung Kim [16] perform angle control using PID, but cart position was unmanageable. PID controller is used as a comparison to neural network for angle and position control of mobile robot.

Gonzalez, Al-varado, Pena [18] [11] combine two linear controllers for construction of low cost balancing robot. A 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.

## IV. Robot Dynamics

A wheeled cart and inverted pendulum system is depicted in Fig. 1. The differential equations were formulated for these degrees of freedom from first principles employing Newton's second law as shown below:

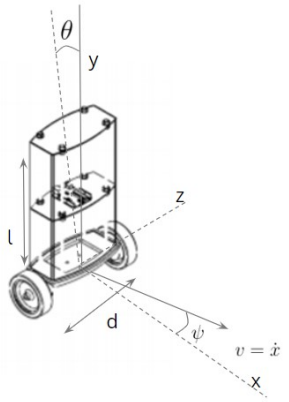


Fig. 1. Model of wheeled inverted pendulum robot

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

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

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

where

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

Table I, defines the parameters of this dynamic model.

TABLE I
THE PARAMETERS OF TWO-WHEELED INVERTED PENDULUM ROBOT

| | | |
|---|---|---|
| $m_{cart}$ | Mass of the cart | 0.493 Kg |
| $m_{pend}$ | Mass of the pendulum | 0.312 Kg |
| $m_{wheel}$ | Mass of the Wheel | 0.050 Kg |
| $I_{pend}$ | Inertia of the pendulum | 0.00024 Kg.m$^2$ |
| $I_{wheel}$ | Inertia of wheel | 0.000074 Kg.m$^2$ |
| $I_{pendY}$ | Inertia of robot about Y-axis | 0.001972 Kg.m$^2$ |
| $l$ | Length of the pendulum | 0.04 m |
| f | friction constant | 0.01 N.s/m |
| $k_t$ | Motor torque constant | 0.11 N.m/A |
| R | Motor resistance | 10 ohm |
| r | Wheel radius | 0.0335 m |
| d | Distance between two wheel centers | 0.165 m |

**Linearized Dynamics:**

The model can be broken up into two disjoint systems, one associated with the position, velocity, tilt and rate of tilt and another for the heading angle and rate of change of heading angle.

The linearized model of these systems around the equilibrium point $x = \dot{x} = 0$, $\theta = \dot{\theta} = 0$ and $\psi = \dot{\psi} = 0$ expressed in the following state space form:

**Velocity and Tilt model**

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

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

$$States = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T \quad (4)$$

where,

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

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

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

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

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

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

**Heading Angle model**

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

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

$$States = \begin{bmatrix} \psi & \dot{\psi} \end{bmatrix}^T \tag{5}$$

where,

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

## V. Full Robot MPC

### A. MPC feedback controller for voltage output

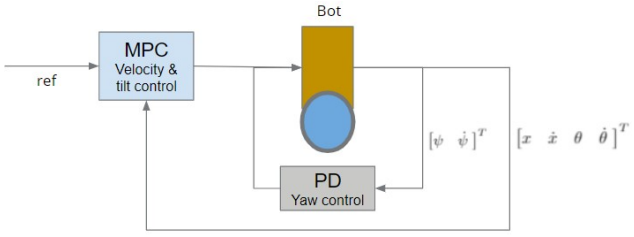


Fig. 2. MPC with voltage output control architecture

The MPC controller was formulated to control the velocity, tilt and rate of tilt while a separate PD controller was used to control the heading angle. The linearized model dynamics were obtained after discretizing them using the c2d function in MATLAB. This control loop is illustrated in Fig. 2. Position was calculated from the velocity, heading angle and previous position at each time step. The MPC controller was designed to track a reference velocity, which was taken as the norm of consecutive way-points from a predefined track. Based on the reference, the MPC controller provides the voltage to reach the desired velocity. There were hard-constraints on the maximum tilt of the robot and the limits of the actuation [-12 V,12 V]. This was done to ensure that the robot doesn't move beyond its controllable domain.

A PD controller was used to correct for heading angle.

As illustrated in Fig. 3 for position, Fig 4 for control effort and Fig. 5 for tilt response, the linearized model of robot under the MPC (Velocity/Tilt) and PD (Yaw) controllers performed satisfactorily in MATLAB simulation. It was able to track the reference trajectory. Further experimentation highlighted that

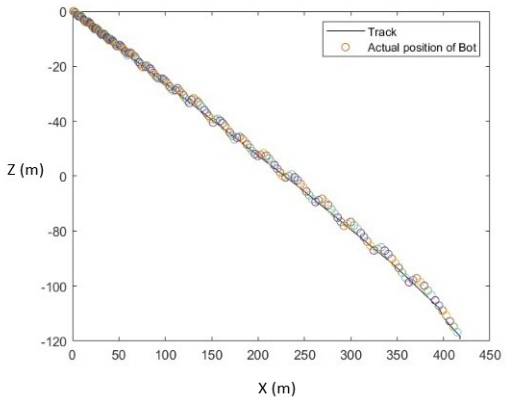


Fig. 3. Position tracking (Top View) with MPC for voltage output

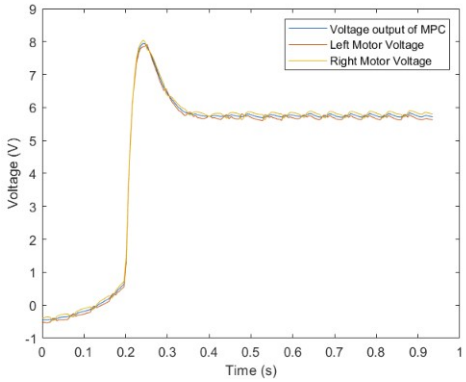


Fig. 4. Control Effort provided by MPC

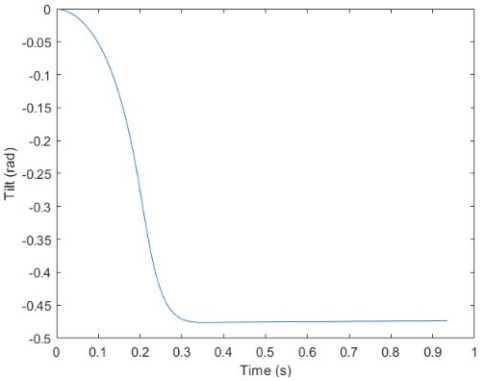


Fig. 5. Tilt of the robot

the linearized model failed to track the reference trajectory under sharp turns. Tuning the Q matrix for better velocity tracking led to poor balancing, with the robot eventually falling out of the feasible constraints region. This happens even under soft constraints on tilt angle wherein after a certain amount of time the robot moved beyond controllable limits of tilt correction and lost balance.

These results are further highlighted and discussed under the Results & Discussions section.

*B. MPC feedforward reference tracking controller with inner loop PD controller*

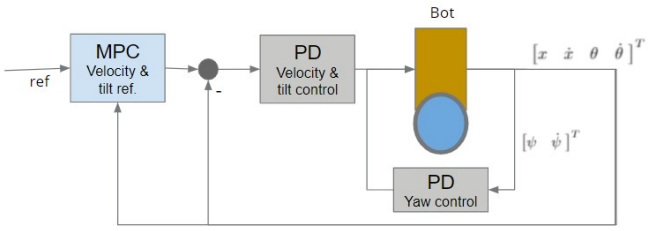


Fig. 6. MPC for reference output control architecture

As the Tumbller had well optimized PD controllers for Position, Tilt and Yaw control, it was sensible to use an MPC control to provide those with optimal references to track. This ensured that the MPC provided the PD controllers with references that had been calculated for constrained quadratic optimization.

**Position and Tilt of robot (No turn)**
This case was formulated for testing the integration of MPC and the PD controller for position and tilt control. The turning PD controller had been omitted to simplify the analysis and verification. In that case the PD controller's gains were derived from the Tumbller's source code.

Position - $K_p = -0.26 \quad K_d = -10$
Tilt - $K_p = 55 \quad K_d = 0.75$

Modified State space for MPC:
$A_d = A - BK$
$B_d = BK$
$X_{K+1} = A_d X_K + B_d \, r_{mpc}$

Based on this formulation the response of the linearized model was simulated in MATLAB. The reference given to the MATLAB was a square wave for position, the norm of subsequent waypoints as reference for velocity and zero for tilt and rate of tilt. There are hard constraints on the Control Voltage to limit it within [-12 V,12 V] and on tilt angle to constrain it between [-0.15 rads,0.15 rads]. The optimized reference output by the MPC was passed to the PD controllers and the response of the system have been illustrated in Fig. 7, 8 and 9.

**Full-robot control**

This model builds on the Position and Tilt only control by adding an additional PD control for heading. The MPC part is the same as the previous case with the same objective function and constraints. As illustrated in Fig. 6. the MPC block is formulated to provide the reference velocity and angle to the inner PD controller. The heading angle is tracked using a separate PD controller.

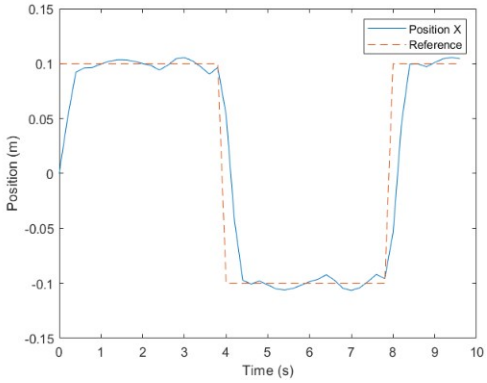


Fig. 7. Position response under MPC for reference output control with only Position and Tilt

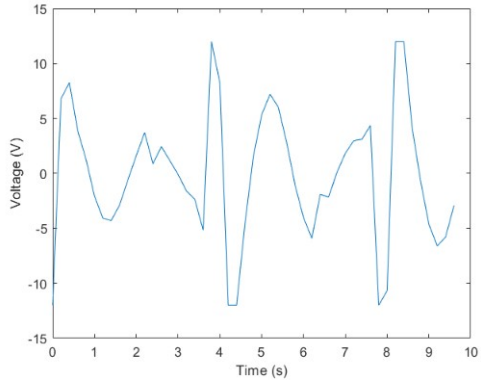


Fig. 8. Voltage response under MPC for reference output control with only Position and Tilt

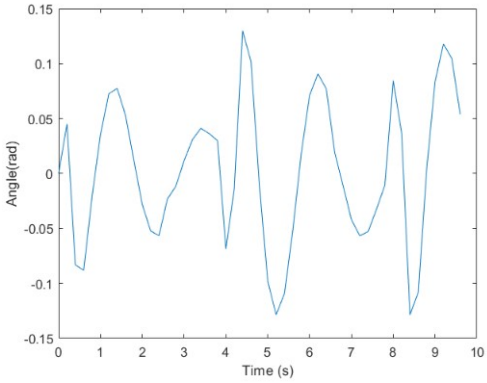


Fig. 9. Tilt of linearized system under MPC for reference output control with only Position and Tilt

This architecture provided the least stable results in terms of decoupling the effects of the MPC and Yaw controllers. Changes in the PD control for heading angle affected the maximum speeds the MPC could provide as reference. This method needs further study and inspection and hence results have not been provided as they did not provide any valuable inference.

## VI. Simulation Models

A two dimensional version of the inverted pendulum system with a cart was considered for the MATLAB simulink model. An advantage of simulation is that it can generate numerical solution to nonlinear equations for which closed form solutions cannot be generated. Non-linear equations mentioned in section IV were used to generate the non-linear Simulink model as shown in Fig. 10. A LQR controller of the linearised system was tested on both linear and non-linear systems.

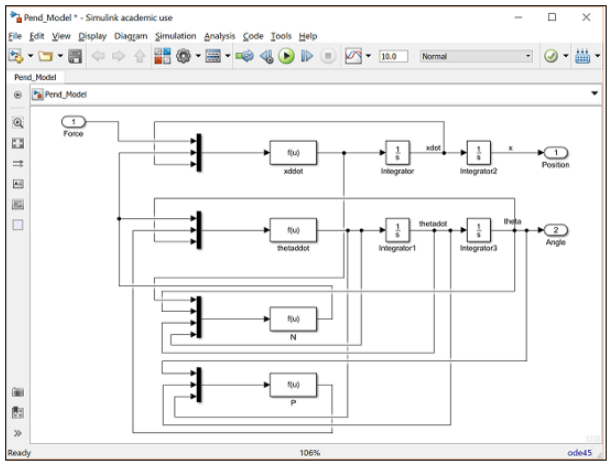


Fig. 10. Non-linear simulink block

The Simscape Inverted pendulum wheeled cart model was built using physical modeling blocks as shown in Fig. 11 and Fig. 12. The blocks in the Simscape library represent actual physical components; therefore, complex multi body dynamic models were built without the need to build mathematical equations from physical principles by applying Newton's laws, which was required in the case presented above. The control input is the force which pushes the cart horizontally and the outputs are the angular position of the pendulum and the position of the cart and their derivatives. Both models mentioned in section[IV] A & B were implemented in Simscape and the results are discussed in section[VIII].

## VII. Hardware

### A. Overview

This project uses an ELEGOO Tumbller Self-Balancing Robot Car Kit as the primary hardware. Out of the box, the Tumbller consists of a MPU 6050 (3 axis gyroscope + 3-axis accelerometer), an ultrasonic sensor, a motor driver, an expansion board and an Arduino Nano microcontroller. In order to achieve all the objectives of this project, additional hardware was added to the standard Tumbller. A Raspberry Pi 3 was added for seamless communication between the robot and a laptop which was running MPC control on MATLAB. A buck converter was used to step down the voltage of the battery from 6.4 V to 5 V required by the Raspberry Pi. A

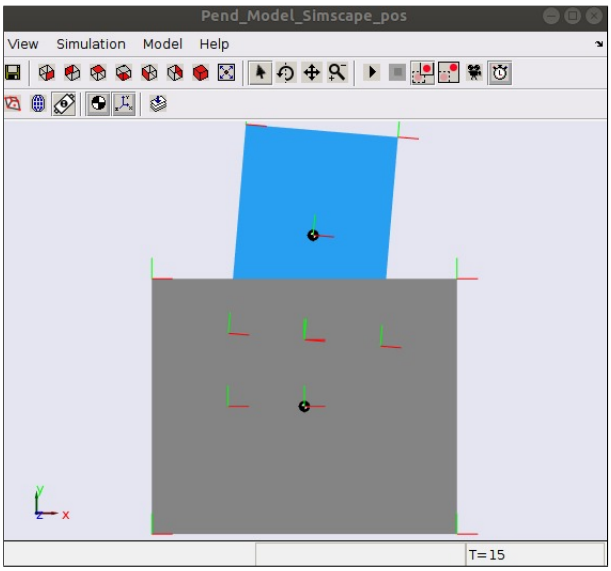


Fig. 11. Simscape Pendulum Model

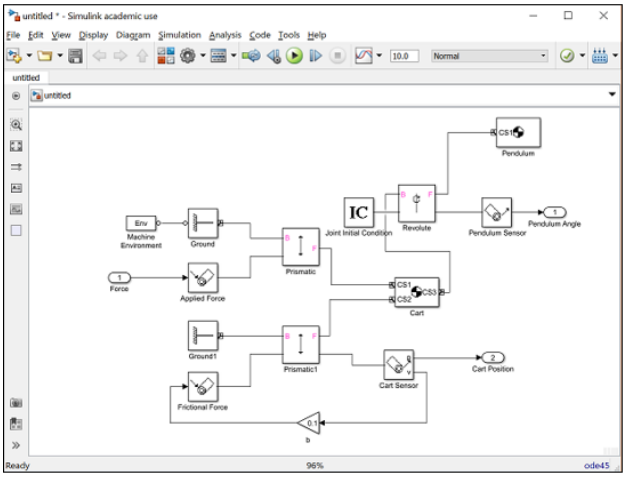


Fig. 12. Non-linear simscape block

Pixy 2 camera was added to obtain ground truth data for implementing line tracking using MPC control on the robot. An additional Arduino Nano was added to act as a slave and send ground truth data obtained from the Pixy to the master Nano which was controlling the robot. Final configuration of the robot is depicted in Fig. 13.

### B. Communication

Robot control for line tracking using Pixy and lime tracking using MPC + Pixy, required the use of two different communication strategies as outlined below.

1) **Line Tracking using Pixy 2 Camera**
In this strategy, Fig. 14, the PixyCam sends a single coordinate (signed 8 bit integer) using ICSP to the slave Nano. This was sent to the master Nano over I2C communication for processing. The Raspberry Pi sends

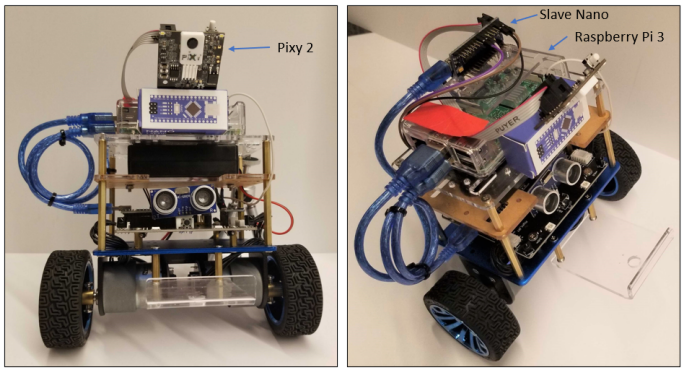


Fig. 13. Hardware

commands from a laptop to the master Nano using serial communication.

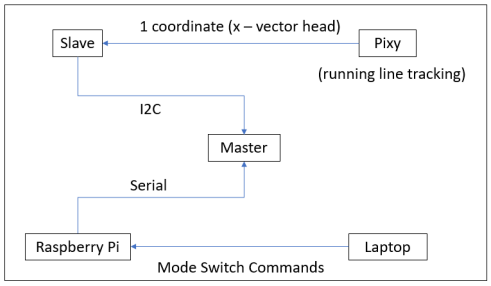


Fig. 14. Communication (Line Tracking)

2) **Line Tracking using MPC + Pixy 2 Camera**
In contrast to the above strategy, PixyCam sends six coordinates (signed 8 bit integers) using ICSP to the slave Nano. This was sent to the Raspberry Pi as a string over serial communication. Similarly, the master Nano sends current measured states to the Raspberry Pi as a string over serial communication. The Raspberry Pi concatenates the two incoming strings and sends it to MATLAB over WiFi for processing. Control inputs calculated using MPC were sent to the Raspberry Pi over WiFi. These were then sent to the master Nano over serial communication. The communication flowchart is depicted in Fig. 15.

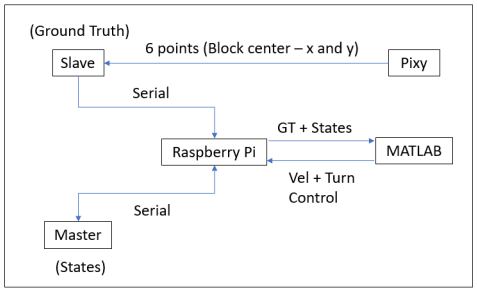


Fig. 15. Communication (Line Tracking + MPC)

## C. Implementation

The modified Tumbller in this project can be operated in four different modes as described below.

1) **Manual Mode**
In this mode W-S-A-D keys are used to move the robot forward, backward, left and right respectively. The key Q is used to cancel the move commands which results in the robot balancing in its current position. These commands are sent manually from the laptop to the Raspberry Pi over WiFi and then to the master Nano over serial. Based on the input received, various motion modes of the robot are triggered which in turn initialize proportional multipliers for forward and turning control. These are then used in the inner loop PD controller which controls the motor PWM. As a result, the robot balances and moves as instructed simultaneously.

2) **Line Tracking Mode**
In this mode, the Pixy runs in its line tracking mode while tracking a solid line, Fig. 17. This mode generates a vector Fig. 17 which indicates the direction in which the robot should move in order to successfully follow the desired path. The slave Nano reads the x coordinate of the head of the this vector and sends it over to the master using I2C protocol. The default forward speed of the robot is set to 5 in this mode. The turn speed of the robot is set proportional to the required turn angle. Fig. 16 shows how Pixy perceives the coordinates of the vector. To determine the required turn angle, the

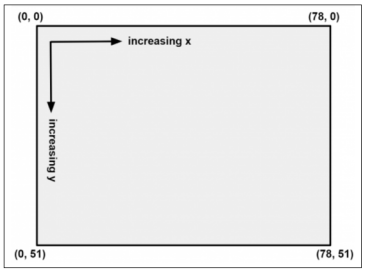


Fig. 16. Coordinates from Pixy's Perspective (Line tracking)

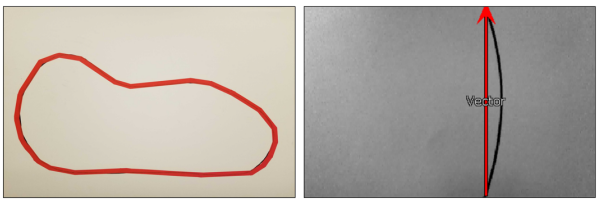


Fig. 17. Continuous Line Tracking Track (left) and Vector Generated by Algorithm (right)

relative position of the robot is determined in respect to the center-line (39) of the Pixy's field of view. If the robot is on the left of the center-line (under 34), the controller turns the robot to the right. Similarly, if the

robot is on the right of the center-line (over 44), the controller turns the robot to the left. If the robot is very close to the center-line (34-44), it will move forward. If the Pixy looses the line, then the controller uses the last value of x to determine turn direction. This ensures that the robot will re-acquire the line if it looses its track.

3) **Line Tracking Mode + MPC**
In this mode, the Pixy runs in its color coded blocks algorithm while tracking a dashed line, Fig. 19. This mode identifies multiple blocks of the line by their color Fig. 19 and returns the x and y coordinates of their centroids. Fig. 18 depicts the coordinates of the blocks as perceived by Pixy. Since, the robot oscillates slightly

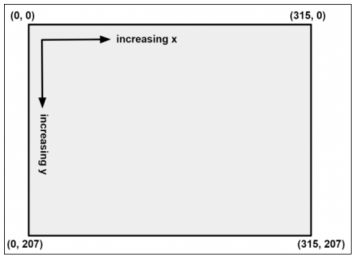


Fig. 18. Coordinates from Pixy's Perspective (Color Coded Blocks)

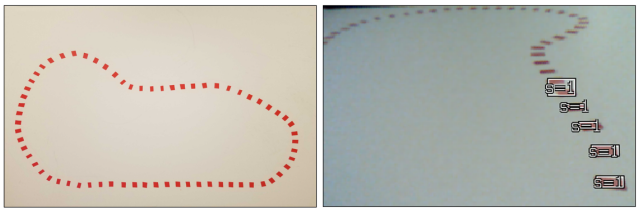

Fig. 19. Discrete Line Tracking Track (left) and Blocks Generated by Algorithm (right)

while balancing, the camera's field of view changes. This will in turn change the mapping of the line blocks to local coordinates. To fix this issue all dashes are evenly spaced and have the same length. This allows us to compensate for the oscillations with good accuracy in MATLAB. The obtained coordinates are sent to the Raspberry Pi as a string via the slave Nano over serial. Raspberry Pi also receives current state estimates over serial from the master Nano as another string. These two incoming strings are concatenated by the Raspberry Pi and sent to MATLAB over WiFi. MATLAB parses these strings one character at a time and stores the inputs in their respective variables. These are used by the MPC to find both speed and turn control inputs. These inputs are sent to the master Nano over serial by the Raspberry Pi.

4) **Follower and Obstacle Detection**
This mode uses the standard ultrasonic sensor provided with Tumbller. The lead robot is controlled manually using a laptop. It uses the ultrasonic sensor to maintain at least 35 cm between the robot and any obstacle in front of it. If the sensor detects an obstacle, it will automatically switch the robot to balance mode and ignore any motion input other than reverse given by the user. The follower robot uses a simplified version of our code. If the ultrasonic sensor detects a lead robot, the follower robot reads the infrared sensors to determine the direction of the lead robot. It uses this direction to steer towards the lead robot. The follower keeps moving towards the lead robot will it is within 5 cm from the lead robot. It will then keep following the lead robot while maintaining a minimum of 5 cm distance from it.

## VIII. Results Discussions

Two state-spaces have been used for MPC formulation: $[x \;\; \dot{x} \;\; \theta \;\; \dot{\theta}]$ and $[\dot{x} \;\; \theta \;\; \dot{\theta}]$.

The MPC with position works on the complete state space. However, since the position is not directly affected by the control input, it is likely that controlling the velocity and then integrating that to get the position at each time step would be a more viable option. Hence both the methods were experimented and hypotheses verified.

MPC feedback controller with nonlinear Simscape model was tested and fine tuned for two linearized MPC models as mentioned in section V A. First model's state space included velocity, tilt and rate of tilt. The cost function is defined with R = 0.1 and Q = [200 0 0 ; 0 1000 0;0 0 1]; the results of velocity tracking are shown in Fig 20 and the control output of MPC is shown in Fig 21

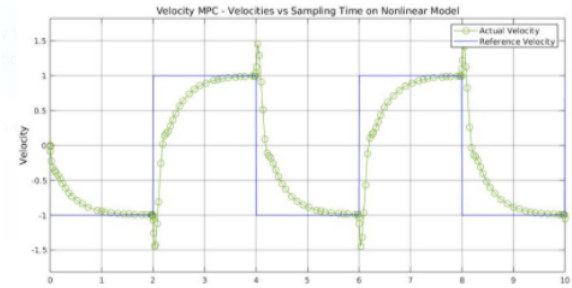


Fig. 20. Actual Velocity Vs Reference Velocity

In the second model we consider position as an additional state and formulate a new linearised MPC controller. The cost function is defined with R = 100 and Q = [100 0 0 0; 0 1000 0 0;0 0 60000 0;0 0 0 1]; the results of position tracking is shown in Fig 22 while velocity tracking is shown in Fig 23 and the control output of MPC is shown in Fig 24

It was much easier to tune MPC controller parameters with the velocity model and then using those initial results as a reference, tune the second model with position included in the reference model. This was one of the greatest advantages

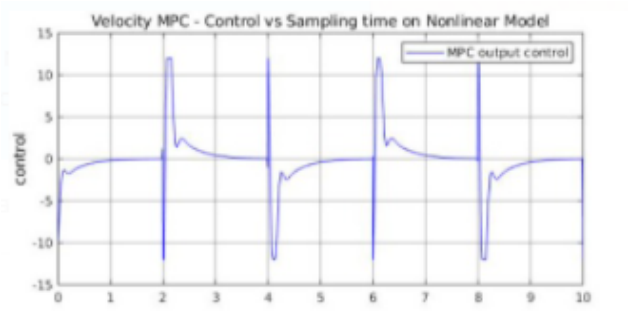


Fig. 21. MPC Output Control

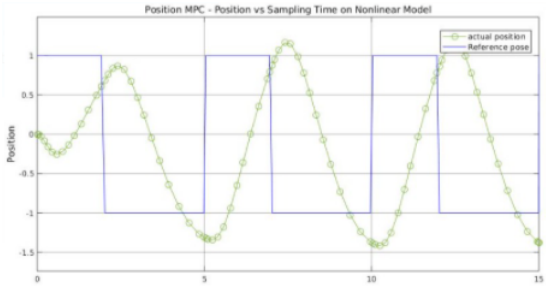


Fig. 22. Actual Position Vs Reference Position

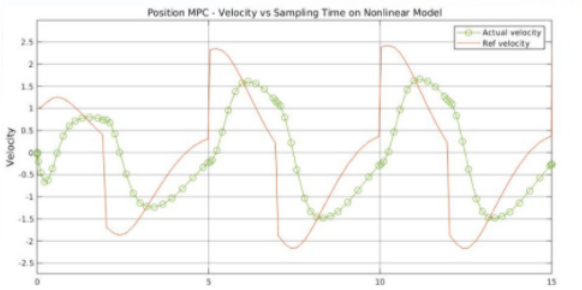


Fig. 23. Actual Velocity Vs Reference Velocity

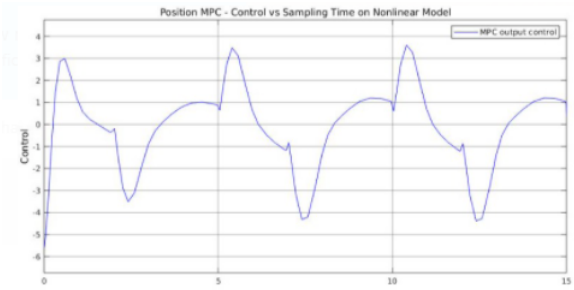


Fig. 24. MPC Output Control

of MPC that we leveraged by using two state space models. Based on the above results, we can see that both models perform very well when tracking position and velocity. Also the control constraints and output tilt angle constraints are very kept well within their respective range.

MPC feedforward control with inner loop PD control as mentioned in section VII C nonlinear Simscape model was tested and fine tuned for two linearized MPC models as mentioned above. In the first model with state space variables as velocity, tilt and rate of tilt and inner loop PD controller gains as K = [-23.3, 790.4, 14.5], a cost function with R = 0.1 and Q = [100 0 0 ; 0 10000 0;0 0 1]; the results of velocity tracking are shown in Fig 26 and the control output of MPC is shown in Fig 25

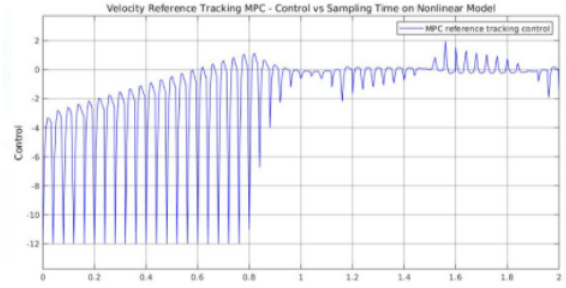


Fig. 25. MPC Reference Tracking Control

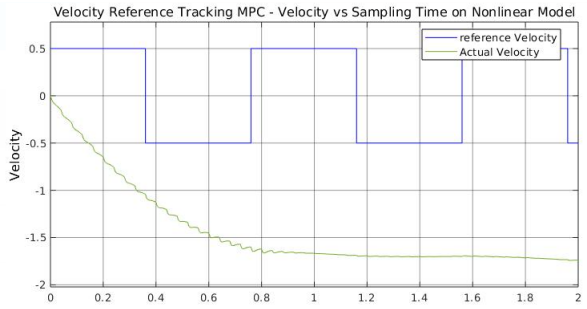


Fig. 26. Reference Velocity Vs Actual Velocity

In the second model we consider position as an additional state and formulate linearised MPC. Inner loop PD control has K = [-1 -23.3, 790.4, 14.5]; In cost function with R = 100 and Q = [100 0 0 0; 0 1000 0 0;0 0 60000 0;0 0 0 1]; the results of position tracking are shown in Fig 27 while velocity tracking is shown in Fig 28 and the control output of MPC is shown in Fig 29

Feedforward Reference Tracking MPC with inner PD controller in both the models tries to always reach a constant velocity for any given reference as can be seen in Fig 26 & 28. However, the control constraints and output tilt angle constraints are very well within the range. Hence, for reference tracking even though the robot managed to maintain balance, it struggled at following the reference position and velocity. The jittery control behaviour probably stemmed from fixed frequency noise in the system, stemming from the hardware or communication protocol.

With these Simulations as a starting point, PD control constants were incorporated into the hardware for Line Tracking implementations with and without MPCs. We have observed

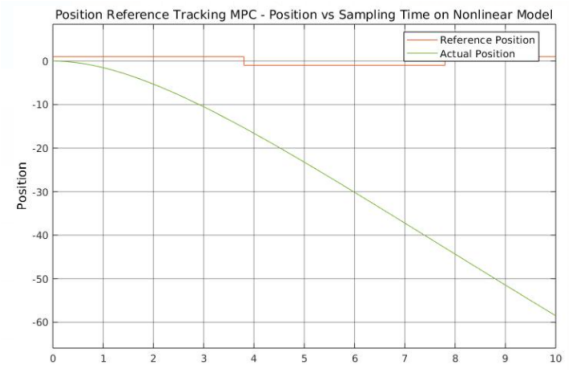


Fig. 27. Position Reference Tacking MPC - Position vs Sampling Time

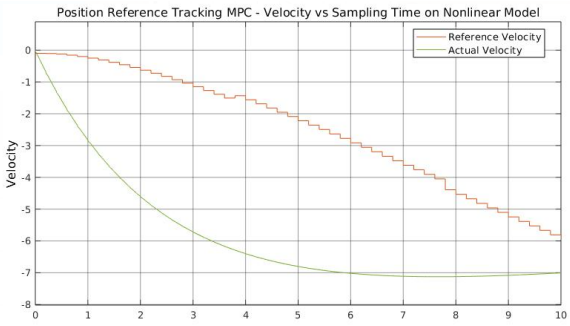


Fig. 28. Position Reference Tacking MPC - Velocity vs Sampling Time

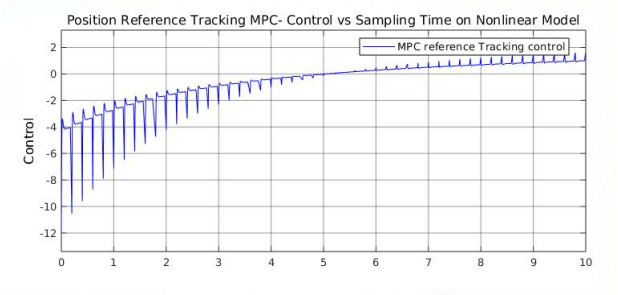


Fig. 29. Control Vs Sampling Time

similar results in Hardware as well which are discussed in section IX and X.

## IX. Challenges

### A. Hardware

The hardware of the Tumbller robot was very inconsistent, the Arduino Nano supplied with the kit lost the ability to upload to it within the first few weeks. The replacement Lafvin Ardunio Nano's also had quality control issues as many of them were faulty, they would either not accept incoming serial data or not accept complied code. The motor encoder readouts were inconsistent as shown by oscilloscope reading conducted by other students, this manifested itself in the robot by steering the robot in a particular direction even when the PD controller for steering was set to zero.

### B. Parsing data in MATLAB

MATLAB receives data from Raspberry Pi as string. Each string consists of data separated by a delimiter ($) and a different delimiter (#) which marks the end of the string. There is a significant delay during parsing of these strings in MATLAB. This occurs as MATLAB has to read and process these strings character by character. This processing is done using while loops which store data in a temporary buffer. Once MATLAB reads the string end delimiter (#), we use the fscan function in MATLAB to separate data from delimiters ($). This stores information in a cell array from which data is extracted. This processing leads to around 08 to 1.5 second delay. This creates a significant issue as the states measured seven iterations ago are used to calculate MPC input for the current iteration. This leads to inaccurate tracking.

### C. Communication

As mentioned in the data processing section the team tried to diagnose the long parsing times by changing the communication protocols of the PixyCam2 from ICSP to serial over RX and TX pins, this affected the functionality of the Tumbller as the TX and RX pins are used to set motor direction and could not be changed if the robot was to maintain quadrature of encoders. Direct communication of the PixyCam2 to the master Arduino Nano over I2C did not work even though the communication protocol was listed as supported by the manufacturer. This drove the decision to use an additional slave Ardunio Nano. Serial communication from the slave Arduino Nano to the master was also problematic due to the pin changing direction of the motors as mentioned above. Serial over GPIO directly into the Raspberry Pi was also tried however there were issues reading it into MATLAB.

## X. Conclusion

In conclusion the Tumbller was able to balance and follow lines with a PD controller effectively. For the final demo the team was able to incorporate obstacle detection and a follower robot. In terms of MPC the robot was able to track a line of fixed distance. Full robot MPC was not achieved due to the communication delays encountered in parsing the incoming data stream. From the limited testing performed due to restricted on campus access the robot had a wide range of functionality. This aligned with the team's goal of making a scaled version of a self balancing multi-agent robot system that is able to function in as an automated guided vehicle.

## XI. Future Work

In retrospect, based on the communication delays observed in running the paring scripts on MATLAB, the controller would have been better implemented if run locally on the Raspberry Pi either through MATLAB or an embedded script derived from MATLAB Coder. By generating C and C++ code for the Raspberry Pi platform along with libraries for the IMU and Tumbller source code, a standalone version of the program could be generated.

## ACKNOWLEDGMENT

We would like to thank Professor Mark Bedillion, Mechanical Engineering Department, CMU for his guidance and support with this project. We would also like to thank Guru Kaushik Senthil Kumar, TA, 24-774, for his suggestions that resulted in improvements throughout this project.

## REFERENCES

[1] S.-C. Lin and C.-C. Tsai, “Development of a self-balancing human transportation vehicle for the teaching of feedback control”
[2] F. Grasser, A. D’arrigo, S. Colombi, and A. C. Rufer, “Joe: a mobile, inverted pendulum”
[3] A. Y. Ivoilov, V. A. Zhmud, and V. G. Trubin, “The tilt angle estimation in the inverted pendulum stabilization task”
[4] S. Jung and S. S. Kim, “Control experiment of a wheel-driven mobile inverted pendulum using neural network”
[5] C. Gonzalez, I. Alvarado, and D. M. La Peña, “Low cost two-wheels self-balancing robot for control education”
[6] Navid Dini, Vahid Johari Majd ”Model Predictive Control of a Wheeled Inverted Pendulum Robot” Proceedings of the 3rd RSI International Conference on Robotics and Mechatronics October 7-9, 2015, Tehran, Iran
[7] Giacomo Galuppini, Lalo Magni, Davide Martino Raimondo,” Model predictive control of systems with deadzone and saturation ”
[8] Samir Bouzoualegh, El-Hadi Guechi and Ridha Kelaiaia ,”Model Predictive Control of a Differential-Drive Mobile Robot”
[9] Shuyou Yu, Yang Guo, Lingyu Meng, Ting Qu, Hong Chen,”MPC for path Following Problems of Wheeled Mobile Robots”
[10] Kai Zhang, Ruizhen Gao and Jingjun Zhang,”Research on Trajectory Tracking and Obstacle Avoidance of Nonholonomic Mobile Robots in a Dynamic Environment”
[11] Nur Uddin, Teguh Aryo Nugroho, and Wahyu Agung Pramudito, ”Stabilizing Two-Wheeled Robot Using Linear Quadratic Regulator and States Estimation””
[12] Hussein Al Jleilaty, Daniel Asmar, and Naseem Daher, ”Model Reference Adaptive Control of a Two-Wheeled Mobile Robot”
[13] Simulation: https://ctms.engin.umich.edu/CTMS/index.php?example =InvertedPendulumsection=SimulinkModeling
[14] https://www.mathworks.com/help/mpc/examples/swing-up-control-of-a-pendulum-using-nonlinear-model-predictive-control.html
[15] https://www.mathworks.com/help/simulink/ug/creating-an-example-model-that-uses-a-matlab-function-block.html
[16] ”wiki:v2:pixy_regular_quick_start”, March,19 2020. [Online]. Available: https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:pixy_regular _quick_start
[17] ”Line tracking for line-following,” March, 07 2018. [Online]. Available: https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:line_tracking
[18] ”Color Connected Components,” October, 29 2018. [Online]. Available: https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:color_connected _components
[19] ”Raspberry Pi Support from MATLAB,” [Online]. Available: https://www.mathworks.com/hardware-support/raspberry-pi-matlab.html
[20] ”Putty: SSH using Windows,” [Online]. Available: https://www.raspberrypi.org/documentation/remote-access/ssh /windows.md
[21] ”Arduino: Master Writer/Salve Receiver,” [Online]. Available: https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter