Model Predictive Control for Ball Bouncing on a Robot Arm

Xingze Dai
Mechanical Engineering Department
University of Michigan, Ann Arbor, MI 48109

Supervised by

Andrew Wintenberg[1] Kwesi Rutledge[2] Necmiye Özay[1]

[1] Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor, MI 48109
[2] Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, MA 02139

Honors Capstone Project
Fall 2022 Semester

Contents

Abstract

This project designs and utilizes a model predictive controller to control a robot arm to wield a paddle and send a ball to a certain height by bouncing it on the paddle. Simulations are conducted to realize the project in a virtual environment. A full stack development from algorithm to hardware implementation is also performed.

Keywords: Model predictive control, KINOVA KORTEX, ball bouncing, Drake simulator

1. Introduction

Model predictive control (MPC) as an advanced control theory has gained popularity recently. Compared to classic PID controls, it features predictions over future time horizons. The goal of MPC is to find a sequence of control actions such that the system of interest achieves a desired performance criterion defined by a loss function in a finite time horizon in the future. Ball bouncing is an interesting task for many robotic researchers because of the complex dynamics of bouncing. Studies have occurred in applying MPC to ball-bouncing tasks to take advantage of the MPC to solve discontinuous dynamics[1]. Also, research on MPC stabilizing hybrid dynamic systems emerged, providing another insight into ball bouncing [2]. Marcucci and Tedrake spent great efforts in formulating and resolving MPC in piecewise-affine systems [3]. And in their paper, they also demonstrate the ball-bouncing algorithm and simulation. In this project, we cast chief attention on Marcucci's demonstration. We will build a new MPC algorithm in the Drake simulator based on their implementation to simulate the actuation of the KINOVA robot arm. We will also adapt the algorithm from simulation to hardware application. A vision module in addition to the existing apparatus will be added and tuned for data acquisition. In conclusion, MPC has the potential to be applied to ball-bouncing tasks. Still, technical challenges are pending to be resolved before it can successfully and stably complete the task in real time.

2. Modeling

Analyzing system behaviors with a model enables more precise controls. In this problem, we refer to Marcucci's model studied and implemented in 2D [3].

2.1 Piecewise Affine System

For the purpose of our study, the model predictive controller aims to provide commands to the paddle to complete the task. The ball that bounces on the paddle is the target to be controlled, which is the task. Thus, the combination of the paddle and the ball is considered a single system.

The system is a hybrid dynamic system. It consists of two modes: free-fall kinematics and collision kinematics. When the paddle and the ball are not in contact, the ball should follow free-fall kinematics. When the paddle and the ball are in contact, the ball should follow collision kinematics. There is hardly a single closed-form describing two kinematics other than a group of piecewise equations. Assumptions are made to simplify the study: The paddle has little inertia. The contact surface is non-slip. No disturbance is included in the study by intention.

2.1.1 Free-fall Kinematics

The distance between two objects is defined as the distance between two centers of mass (C.O.M) of these objects. Free-falling is defined such that the distance between the paddle and the ball is larger than a given small enough, positive, real value. This small value slackens the definition of contacting and provides convenience to system discretization. Figure 1 shows an example of the free-falling kinematics of the system where the ball is not in contact with the paddle, adapted from Marcucci's paper. It denotes geometries and dimensions. In the diagram, x_1, x_2 are the ball's positions; x_3 is the ball's rotation; x_4, x_5 are the paddle's positions; m, j, r are the ball's mass, the moment of inertia, and the radius; l, μ are paddle's length and coefficient of restitution; g is the gravity acceleration.

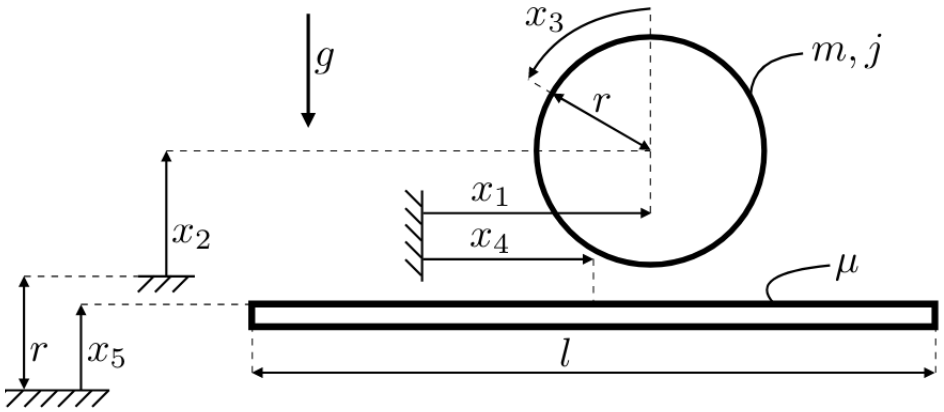


Figure 1. An adapted diagram of the system.

A group of differential equations can be concluded to describe the state space model of the system in Eq. 1.

$$\begin{aligned}
&\dot{x}_1 = \dot{x}_{ball} = v_1 \quad && \dot{x}_4 = \dot{x}_{paddle} = v_4 \quad && \dot{x}_6 = \dot{v}_1 = 0 \quad && \dot{x}_9 = \dot{v}_4 = u_1 \\
&\dot{x}_2 = \dot{y}_{ball} = v_2 \quad && \dot{x}_5 = \dot{y}_{paddle} = v_5 \quad && \dot{x}_7 = \dot{v}_2 = -g \quad && \dot{x}_{10} = \dot{v}_5 = u_2 \\
&\dot{x}_3 = \dot{r}_{ball} = \omega \quad && && \dot{x}_8 = \dot{\omega} = 0 \quad &&
\end{aligned} \tag{1}$$

We can approximate the continuous state space model by a discrete time state space model. Semi-implicit Euler method is applied and ensures the stability of the dynamic system, shown in Eq. 2. h is the discrete time step. u_1 and u_2 are the acceleration of the paddle in horizontal and vertical directions.

$$\begin{aligned}
&x_{1+} = x_1 + hx_{6+} \quad && x_{4+} = x_4 + hx_{9+} \quad && x_{6+} = x_6 \quad && x_{9+} = x_9 + hu_1 \\
&x_{2+} = x_2 + hx_{7+} \quad && x_{5+} = x_5 + hx_{10+} \quad && x_{7+} = x_7 - hg \quad && x_{10+} = x_{10} + hu_2 \\
&x_{3+} = x_3 + hx_{8+} \quad && && x_{8+} = x_8 \quad &&
\end{aligned} \tag{2}$$

Therefore, the kinematics for the free-falling condition is achieved.

2.1.2 Collision Kinematics

Contrary to the definition of free-falling, collision is defined such that the distance between the paddle and the ball is smaller than a given small, positive, real value. It is convenient to assume that collision happens regardless of the true value of distance if the collision definition is satisfied. Figure 2 describes the collision between the ball and the paddle.

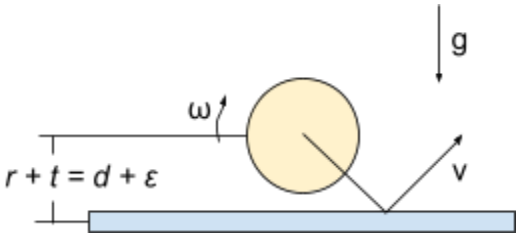


Figure 2. A kinematic diagram of collision.

In this project, the collision model is dependent on the force. By the no-slip surface assumption, a force balance equation can be solved and the kinematics is updated based on the force. A group of differential equations can be concluded to describe this system in Eq. 3.

$$\begin{array}{llll} \dot{x}_1 = \dot{x}_{ball} = v_1 & \dot{x}_4 = \dot{x}_{paddle} = v_4 & \dot{x}_6 = \dot{v}_1 = f_{pt}/m & \dot{x}_9 = \dot{v}_4 = u_1 \\ \dot{x}_2 = \dot{y}_{ball} = v_2 & \dot{x}_5 = \dot{y}_{paddle} = v_5 & \dot{x}_7 = \dot{v}_2 = f_{pn}/m - g & \dot{x}_{10} = \dot{v}_5 = u_2 \\ \dot{x}_3 = \dot{r}_{ball} = \omega & & \dot{x}_8 = \dot{\omega} = r \cdot f_{pt}/j & \end{array} \quad (1)$$

Similarly, backward discretization is applied to approach the differential equation numerically, shown in Eq. 4. h is the discrete time step.

$$\begin{array}{llll} x_{1+} = x_1 + hx_{6+} & x_{4+} = x_4 + hx_{9+} & x_{6+} = x_6 + hf_{pt}/m & x_{9+} = x_9 + hu_1 \\ x_{2+} = x_2 + hx_{7+} & x_{5+} = x_5 + hx_{10+} & x_{7+} = x_7 + hf_{pn}/m - hg & x_{10+} = x_{10} + hu_2 \\ x_{3+} = x_3 + hx_{8+} & & x_{8+} = x_8 + rhf_{pt}/j & \end{array} \quad (4)$$

Therefore, the kinematics of the collision is achieved.

2.1.3 Hybrid kinematic system

Regarding the efforts from previous sections, a discrete affine system is gained in Eq. 4. By switching on and off the force terms, f_{pt} and f_{pn}, the affine system is piecewise.

2.2 Formulation of Model Predictive Control

Loss functions are associated with the performance of a system. They take arguments (parameters, states, inputs, etc.) from the system and compute a loss from optimality. If the system is away from the optimum performance, the loss will increase. The better the system performance is, the less the loss is.

It is practical to rely on the discretized piecewise affine system concluded in the previous section and predict the next-step states of the system based on the current states and inputs. Given the prediction time horizon, a sequence of states and inputs can be recursively generated. To evaluate the performance of this state-input sequence, a quadratic cost function in Eq. 5 can be drafted where J is the loss, and N is the time horizon. P, Q, R are weight matrices with $P \succeq 0$, $Q \succeq 0$, and $R > 0$.

$$J_0(x(0), U_0) = p(x_N) + \sum_{k=0}^{N-1} q(x_k, u_k) \tag{5}$$

Optimization is a program to find the best solution such that the loss function is minimized. Model Predictive Control is, therefore, a control dedicated to minimizing the loss of the states from the optimal in a time horizon. In this project, it is described by a Constrained Finite Time Optimal Control problem (CFTOC), written in Eq. 6, where $\mathcal{X}$, $\mathcal{U}$, $\mathcal{X}_\mathrm{f}$ are polyhedral regions [4].

$$\begin{aligned} J_0^*(x(0)) = \quad & \min_{U_0} \quad && J_0(x(0), U_0) \\ & \text{subj. to} && x_{k+1} = Ax_k + Bu_k, \ k = 0, \dots, N-1 \\ & && x_k \in \mathcal{X}, \ u_k \in \mathcal{U}, \ k = 0, \dots, N-1 \\ & && x_N \in \mathcal{X}_f \\ & && x_0 = x(0) \end{aligned} \tag{6}$$

It is acknowledged that the optimal solution of the model predictive control for hybrid systems is difficult to acheive. Therefore, more complex algorithms in Marcucci's implementations were applied to conduct the optimization in the model predictive control. The commercial package, Gurobi, is utilized to solve optimization problems.

3. Simulation

Simulation is written in Python and is developed based on the Drake simulator [5]. Open-source packages—MPC simulation [6], KINOVA hardware interface [7], KINOVA software interface [8], visual fiducial system [9], and so forth—are used during development.

3.1 Architecture

Figure 3 below shows the architecture of the simulation. The architecture is developed mainly using Drake packages. Each block is a class maintaining a single functionality and each arrow indicates data and data flow directions. The Ball block possesses a ball's velocity and position states, and its kinematics. The kinematics is piecewise-affine and the next-step states are predicted by Marcucci's optimization implementation [3]. The Paddle block possesses a paddle's velocity and position states, and its kinematics. It also has two DOFs in acceleration. The MPC Solver accepts states from the ball and paddle and implements the model predictive control. The blue block is a collision detection block that detects whether the ball and paddle collide and outputs a binary result. The Visualizer block synthesizes the simulation context and all state information and visualizes a 3D model on the display.

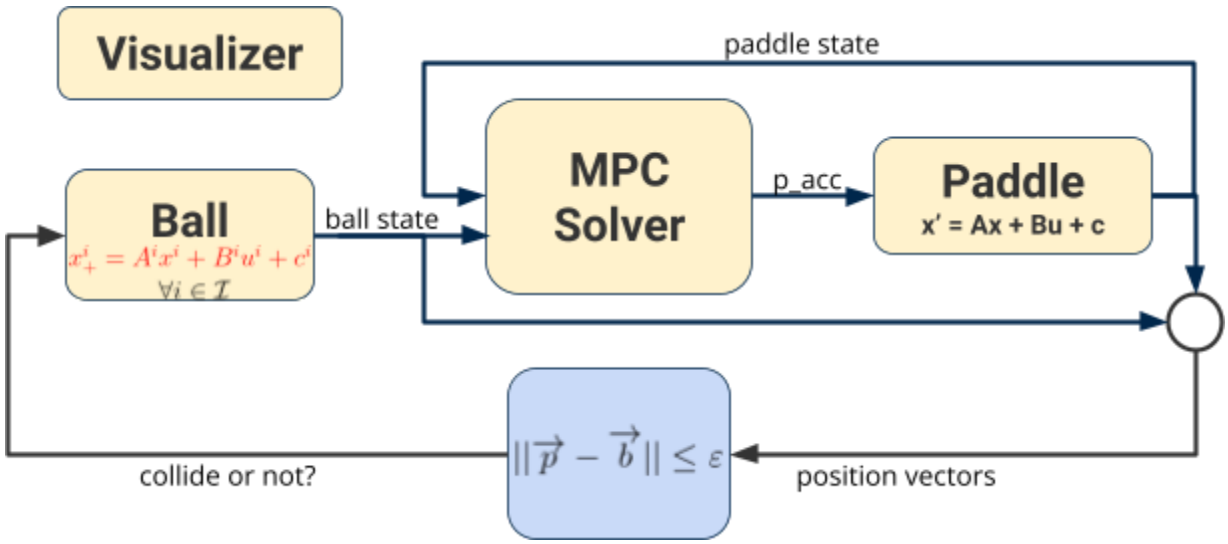


Figure 3. The architecture of the simulation.

The Ball and Paddle with their states are initialized when the simulation starts. At each time step, the ball states are updated by the kinematic system which is initially dominated by free-fall motions. The ball states are then delivered to the MPC Solver. The MPC Solver receives the paddle states and ball states and calculates an optimal 2-DOF input sequence by optimizing the loss function in Eq. 6. The first element in the 2-DOF input sequence is outputted to the Paddle. Given the 2-DOF acceleration inputs, the paddle updates its states based on its kinematic system. The new paddle states are fed back to the MPC Solver. After the Ball and Paddle have been updated, their states are synthesized and sent to the collision detection block. The ball's coordinates and the paddle's coordinates are extracted from the synthesis. The norm of a vector difference is calculated and compared to a preset tolerance; if the norm is larger than the tolerance, no collision occurs, and vice versa. Based on whether the collision occurs, the collision detection block delivers a binary value to the Ball. The Ball receives and stores the binary in its memory which determines the kinematics in the next time step. Finally, the architecture moves to the next discrete time step, and the process described above starts over again.

3.2 Constraints

Constraints define disjunctive polytopes of the state space and the target space in the simulation. The constraints identify the feasibility of the entire system and the range of the optimal solution. Its setup is crucial to the success and the speed of optimization, given that Marcucci has proved the convexity of the formulation. The constraints should also correspond to the real situation. In Figure 4, the ball's (dashed box) and floor's (solid box) positional constraints of state space and target space are plotted. It is intended that the ball's target region is disjoint with the floor's target region so the solution in which the ball sticks to the floor and moves with the floor will be prevented.

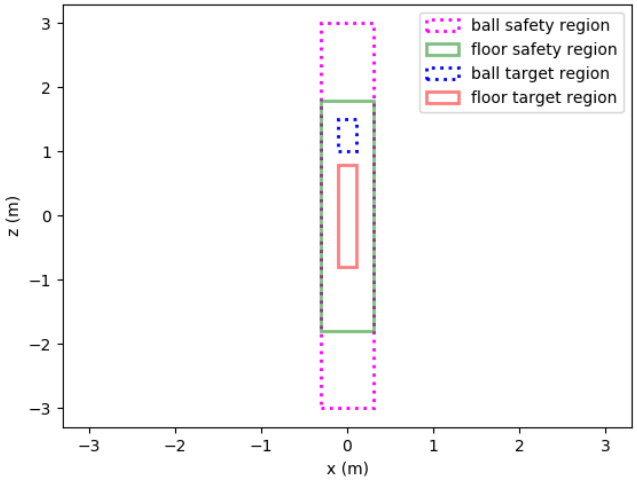


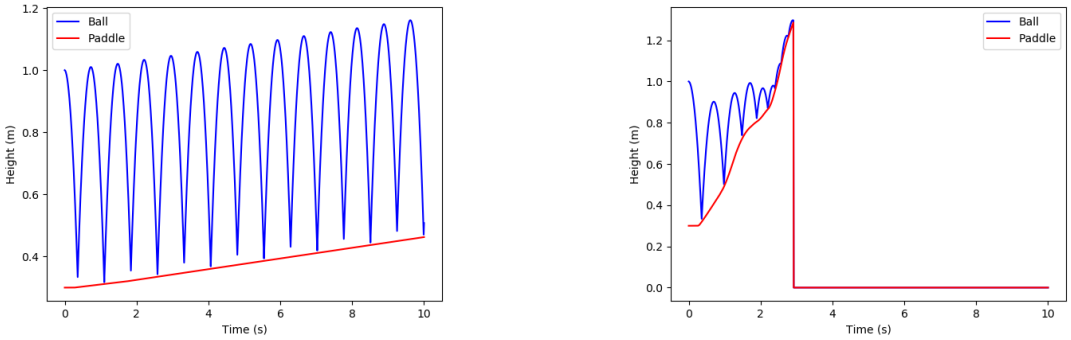
Figure 4. An example of positional constraints.

3.3 Simulation Parameters and Hyperparameters

In simulation, several parameters are available and important to adjust, leading to different results. While other parameters are held constant, simulation parameters are as follows: coefficient of restitution ε, the initial heights of the ball and paddle, and the vertical heights of the target regions. Hyperparameters determine approaches to simulation. They are the size of the prediction step, the number of prediction steps, and the size of the discrete simulation time step. Because the main motion is in the vertical direction despite the 2-dimensional formulation, only the heights in the feasibility space and target set are hyperparameters while the dimensions of those regions in other directions are held constant. The coefficient of restitution is introduced to simulate a realistic collision model and it plays a major role in the feasibility of optimal model predictive control.

3.4 Results

Figure 5 shows two results of the simulation with all but the coefficient of restitution the same. The initial height of the ball is 1.0 and the initial height of the paddle is 0.3. The target regions are set the same as shown in Figure 4. The prediction step size is 0.1. The number of prediction steps is 5. The simulation step size is 0.01. In Figure 5 (a), the coefficient of restitution is 0.99 while in (b) the coefficient is 0.8. When the coefficient of restitution is close to 1.0, which indicates perfect elastic bouncing, the model predictive controller is able to send the bouncing ball and the paddle into the target region. When the coefficient of restitution is 0.8, which implicates a real elastic bouncing energy loss, the model predictive controller is unable to complete the task. Failure in the task is caused by the bouncing energy loss and the ball will stick to the paddle as shown in (b) at about 3 seconds. Because no solution is found by the model predictive controller when the ball stays on the paddle, the heights of the ball and the paddle are zero.



(a) (b)

Figure 5. Two results from the simulation varied by the coefficient of restitution. (a) has a coefficient of restitution of 0.99 and shows a successful control. (b) has a coefficient of restitution of 0.8 and shows that no solution is found at about 3 seconds.

4. Hardware

Hardware implementation is adapted and developed based on the algorithms mentioned above. Hardware includes a Kinova® Gen3 Ultra lightweight robot [10] and an Intel® RealSense™ Depth Camera D415 [11].

4.1 Perception theorem

Compared to simulation in which all data including objects' states are shared with the MPC Solver, the acquisition of the ball's states is required in reality. The MPC Solver can read camera captures to gain the ball's states. Still, a transformation of the ball's states with respect to the world frame is required.

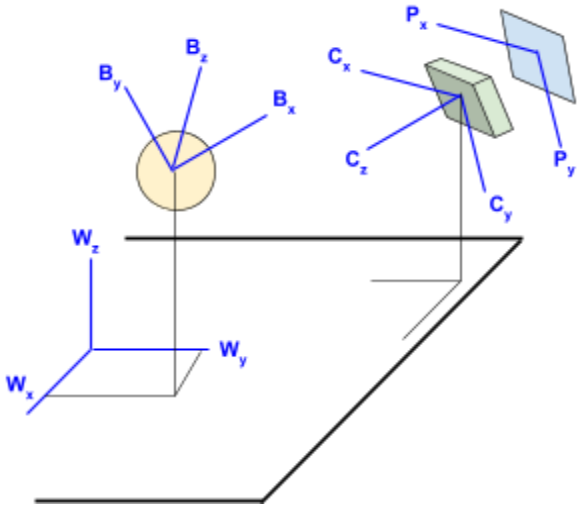


Figure 6. A diagram of the models and frames.

Figure 6 shows a diagram of the perception. *W* represents the world frame. The ball has its orientation frame *B*. The depth camera has an extrinsic pose frame *C*. Also, objects shown on the screen are represented in pixel frame *P*. To get the pose of the ball with respect to the world frame from the vision, a transformation is required.

We follow the convention that the right superscript denotes the "to" frame and the left superscript denotes the "from" frame which is ignored when the transformation is from the world frame. The ball's pose in the world frame is X^B, the camera's pose in the world frame is X^C, the camera's intrinsics is ${}^C X^P$, and the position in the pixel frame is ${}^P X^B$. The transformation of the ball's pose with respect to the pixel frame is calculated by Eq. 7.

$$X^B = X^C {}^C X^P {}^P X^B \tag{7}$$

All transformations will be elaborated in the following sections.

4.2 Camera's Intrinsics Calibration

All cameras used in this project are depth cameras with RGB sensors. One camera is on a tripod (articled static camera) and the other camera is integrated into the KINOVA's wrist. Both cameras are self-calibrated in real time and no distortion occurs due to the pinhole structure. Nevertheless, Intel RealSense provides a convenient SDE to self-calibrate the intrinsics of the static camera.

4.3 Camera's Extrinsics Calibration

Finding the camera's extrinsics resorts to the theorem in Section 4.1 given all other transformations. Apriltags are used to quickly retrieve the pose in the perspective of the depth camera. Two methods are used to find the camera's extrinsics and they can cross-verify the other. The first method is shown in Figure 7.

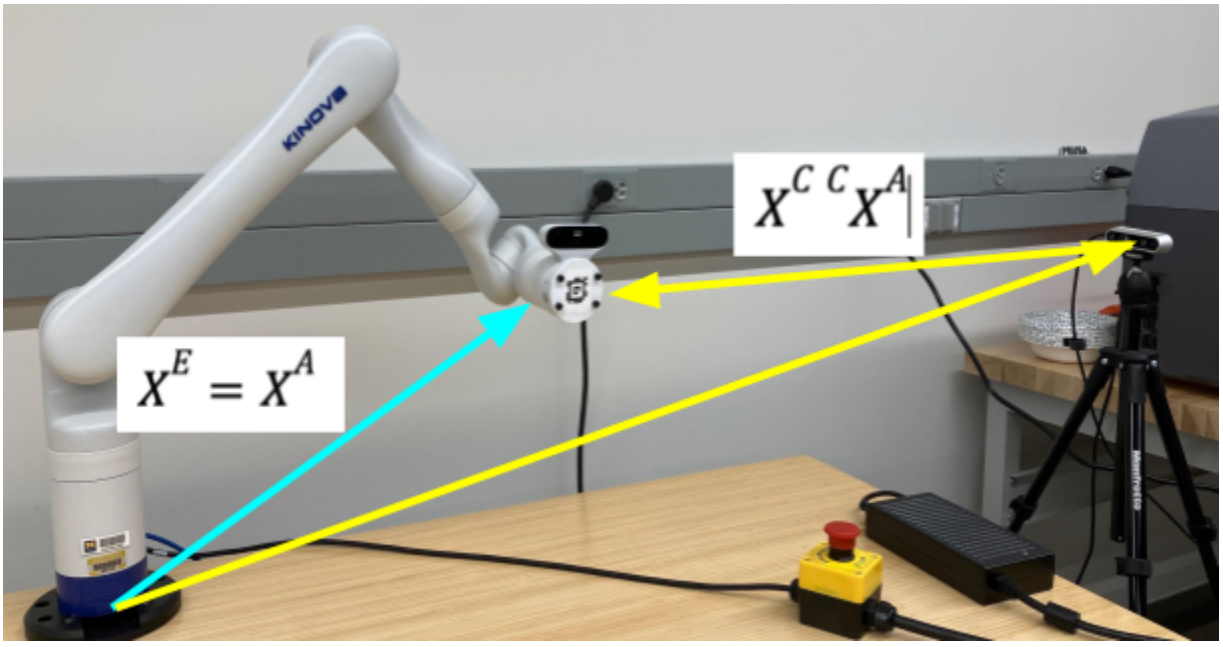


Figure 7. A diagram of the first method using Apriltag and forward kinematics.

In this method, an Apriltag is attached to the end effector of the KINOVA arm. The pose of the end effector can be read from a packaged algorithm from KORTEX which gains the forward kinematics of the end effector. We can assume that the pose of the end effector is the pose of the Apriltag in Eq. 8.

$$X^A = X^E \tag{8}$$

The pose of the Apriltag with respect to the depth camera frame, ${}^C X^A$, is gained by modifying an open example posted by the Apriltag lab [9]. To represent the Apriltag in the world frame, the camera's pose, X^C, is required. Therefore, the pose of the Apriltag can otherwise be represented in Eq. 9.

$$X^A = X^C \, {}^C X^A \tag{9}$$

By equating the Eq. 8 and Eq. 9, the camera's extrinsics can be solved in Eq. 10.

$$X^C = X^E ({}^C X^A)^{-1} \tag{10}$$

The other method is shown in Figure 8 and utilizes the vision module installed on the wrist of the KINOVA arm.

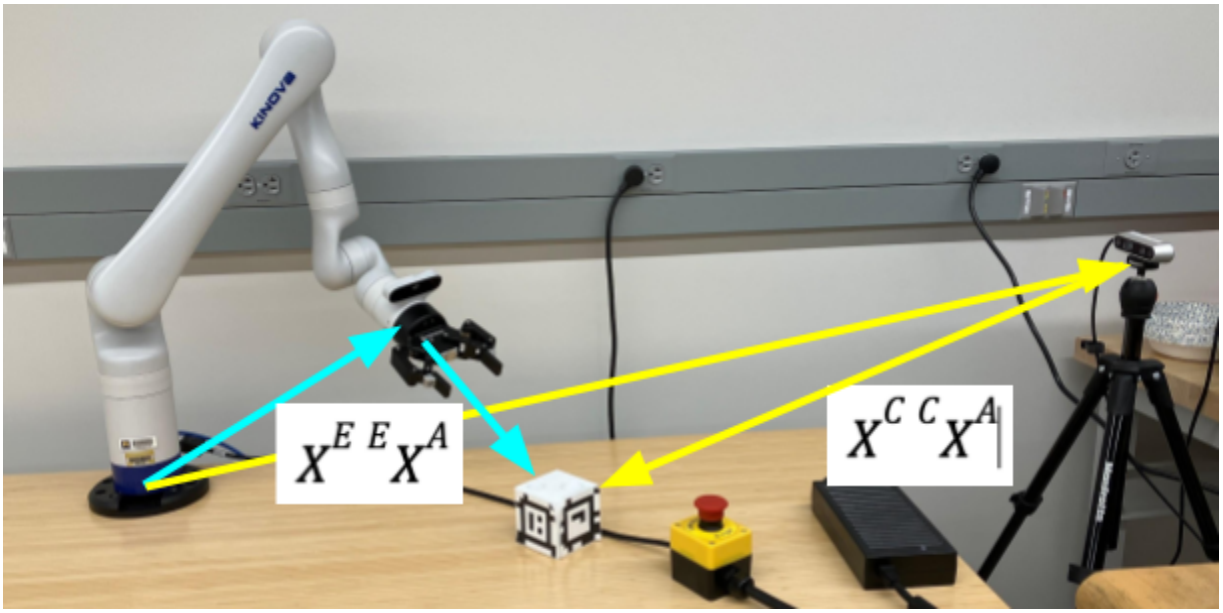


Figure 8. A diagram of the second method using Apriltag and forward kinematics.

In this method, an Apriltag is attached to a cube to perceive. The pose of the end effector, X^E, is read from the forward kinematics and the pose of the Apriltag with respect to the end effector, ${}^E X^A$, is gained. Therefore, the pose of the Apriltag with respect to the world frame can be represented by Eq. 11.

$$X^A = X^E \, {}^E X^A \tag{11}$$

By plugging Eq. 9 into Eq. 11, we can solve for the camera's pose and it is represented in Eq. 12.

$$X^C = X^E \, {}^E X^A ({}^C X^A)^{-1} \tag{12}$$

Both methods render the camera's extrinsics and their results are used to cross-verify one another. A cross-verification example is shown in Figure 9. The distance and orientation difference between the two sets result from errors.

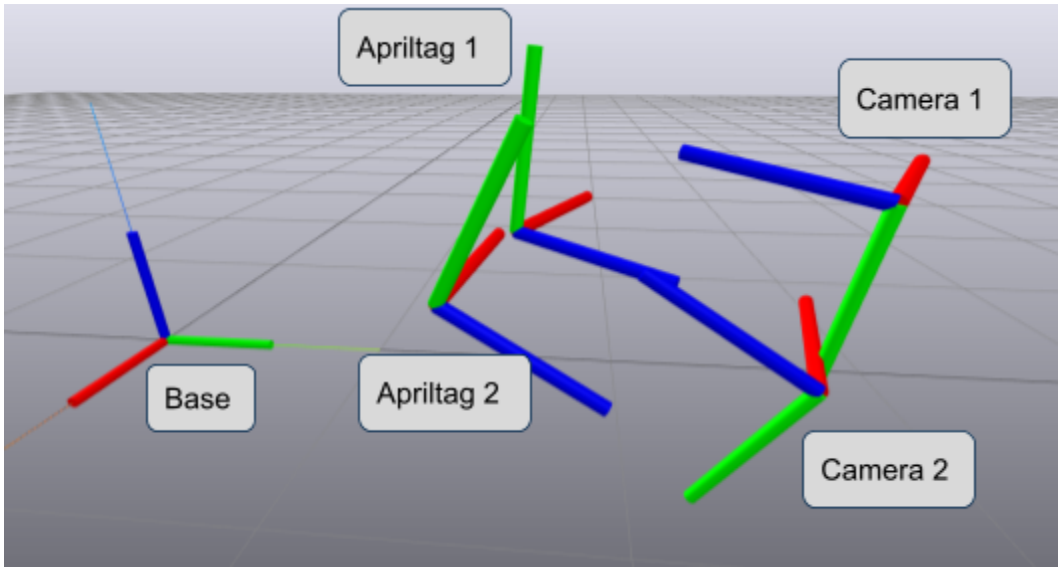


Figure 9. A diagram of cross-verified frame alignments. Errors in the deduced frames exist.

4.3 Ball locating algorithm

The ball locating algorithm is performed on the pixel frame. The background is important to the ball-locating algorithm. A realistic background is preferred to replicate real application situations but visual noise is inevitable and dedicated to being reduced. Color thresholding is applied to narrow down the possibility of the location of the ball. Pixels that have different colors than the ball's color will be blackened. With segments of similar colors in the view, the Hough circle algorithm is applied to find segments that mostly resemble a circle as shown in Figure 10.

(a) (b)

Figure 10. (a) Blackening out unrelated color backgrounds. (b) The ball is found with a yellow dot pointed at its center.

It is assumed that the segment with the highest confidence of resembling a circle is the ball and the center of the found segment is assumed to be the center of the ball. The center's pixel coordinates will be appended by the depth gained from the depth camera, and the positional vector is obtained. Since it is rather difficult to obtain the ball's rotation, the rotation matrix is naïvely assumed to be an identity matrix. Therefore, the pose of the ball with respect to the pixel frame is gained and by Eq. 7, the pose of the ball with respect to the world frame is calculated.

4.4 Architecture

The architecture implemented on the hardware is different from the architecture used in the simulation. The Camera provides the ball's pose to the MPC Solver. The end effector pose is read from the forward kinematics in the backend of the KINOVA arm and is sent to the MPC Solver. The MPC Solver solves the optimal control sequence based on all states and pops out the first 2-dimensional acceleration input to the Controller. The Controller then generates motion commands based on the input and current arm pose and sends them to the backend of the KINOVA arm. The KINOVA arm receives the commands and actuates. At the same time, its pose is sent back to the Controller and the MPC Solver. The diagram of the architecture is shown in Figure 11.

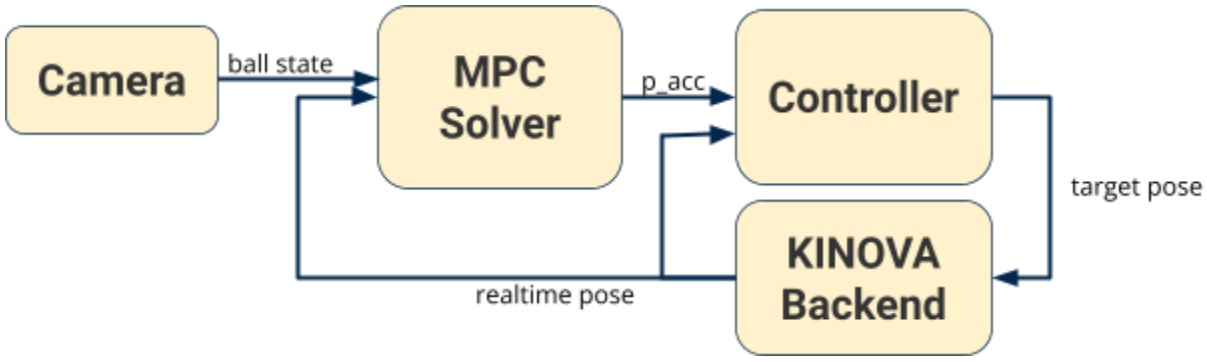
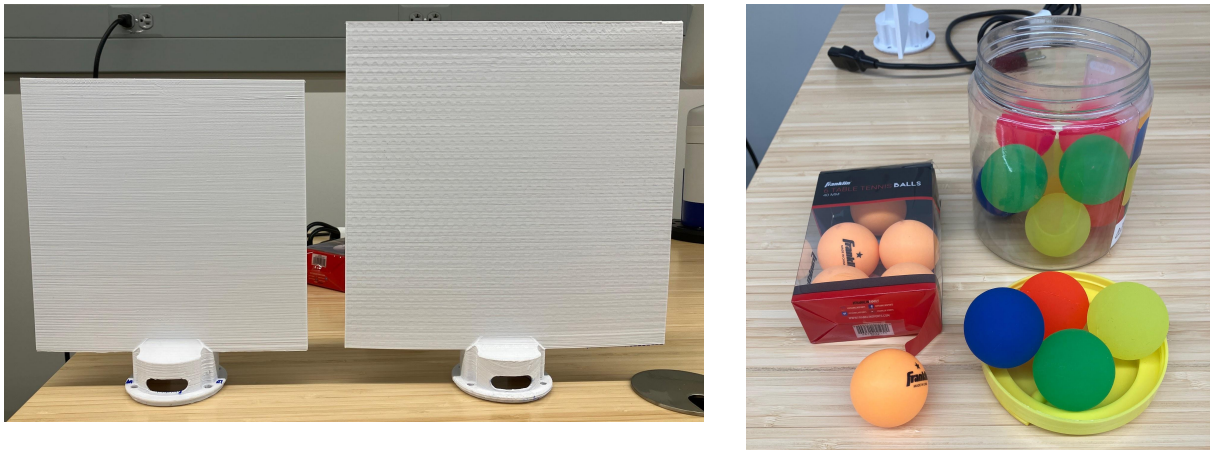


Figure 11. The architecture of the hardware implementation.

4.5 Custom parts

Paddles are customized and manufactured by 3D printing. One is 200 mm in length and the other is 240 mm in length in Figure 12. Their materials are PLA. Their bases are specially designed so that they are adapted to the end effector interface.



(a) (b)

Figure 12. (a) Two 3D printed paddles and (b) ping pong balls and colored rubber balls.

We select ping pong balls and rubber balls in our experiments. They vary in the coefficient of restitution. Also, rubber balls are differently colored which facilitates the color thresholding in Section 4.3.

4.6 Results

The result of the hardware implementation is incomplete. The KINOVA arm is actuated during the first algorithmic cycle but in the next cycle, the MPC Solver reports no feasible solution and aborted the action. Feasibility is delicate and hard to ensure on the hardware. We had attempts in tuning simulation parameters and hyperparameters but little progress was made. Several challenges are pending solutions. First, the error in the coordinate transformation is too large to be ignored. Second, the frame rate of the depth camera is insufficient to clearly capture the ball's high-speed motion, leading to motion blurring. Third, the ball locating algorithm is unable to find a stable trajectory of the ball considering noise and blurring. Fourth, computational constraints may impose difficulties in finding a solution. Observations suggest that in some situations, the MPC Solver takes seconds to find a solution, which is unacceptable in real-time computing. Fifth, some real physical properties will also impede feasibility. As shown in Section 3.4, when the coefficient of restitution is 0.8 which is lower than the perfect elastic coefficient of restitution, the MPC Solver is unable to find a feasible solution to the task. In the actual experiment, the coefficient of restitution for a ball bouncing on the paddle is reasonably lower than the coefficient of restitution in perfectly elastic collisions. Thus, it is highly possible that the MPC Solver still cannot find the solution during the hardware experiment.

5. Conclusion and Future Work

The objective to control a paddle so that it can send a bouncing ball to a certain height using model predictive control is achievable in theory and has been proved in the simulation results. Still, some issues prevent the simulation from realization. For example, the time expense in computing some edge cases is too high to be used in real-time applications. Also, the feasibility is sensitive to the constraints and in many cases, no solution is found, which affects the

performance of the hardware experiment as mentioned in Section 4.6. Attempts to implement hardware tests were made but the robot arm fails to achieve the expected performance. Besides the issues found in the simulation, tuning and debugging apparatus complicates the realization.

In the future, some approaches can be applied to resolve the deficiency. First, a pre-calculated solution set or reduced calling of calculation is anticipated in terms of the heavy computation expense. Second, unless specified by the application environment, new methods in tuning the constraints are needed to explore the boundary of feasibility. Third, an adequate amount of time is required to set and tune the hardware.

Acknowledgments

Great appreciation to Andrew and Kwesi who co-worked and supervised me on this project and provided ideas and generous help. Sincere appreciation to Professor Özay who guided me with care during the research. Also, thanks to other colleagues including Yuhang Mei and Zexiang Liu providing excellent questions and thoughts.

References

[1] Kulchenko, Paul, and Emanuel Todorov. “First-Exit Model Predictive Control of Fast Discontinuous Dynamics: Application to Ball Bouncing.” *2011 IEEE International Conference on Robotics and Automation*, 2011, doi:10.1109/icra.2011.5980196.

[2] Altın, Berk, et al. “A Model Predictive Control Framework for Hybrid Dynamical Systems.” *IFAC-PapersOnLine*, vol. 51, no. 20, 2018, pp. 128–133., doi:10.1016/j.ifacol.2018.11.004.

[3] Marcucci, Tobia, and Russ Tedrake. “Mixed-Integer Formulations for Optimal Control of Piecewise-Affine Systems.” *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, doi:10.1145/3302504.3311801.

[4] Jones, Colin, et al. “Model Predictive Control Part I – Introduction.” *Institut für Automatik ETH Zürich, UC Berkeley, EPFL*, Spring 2015.

[5] “Pydrake.” *Pydrake - Pydrake Documentation*, drake.mit.edu/pydrake/index.html.

[6] M. Tobia, pympc, (2013), GitHub repository, https://github.com/TobiaMarcucci/pympc

[7] KINOVA® KORTEX™ robotic arms, kortex, (2022), GitHub repository, https://github.com/Kinovarobotics/kortex

[8] K. Vince, kinova_drake, (2022), GitHub repository, https://github.com/vincekurtz/kinova_drake

[9] AprilRobotics, apriltag, (2022), GitHub repository, https://github.com/AprilRobotics/apriltag

[10] “Discover Our gen3 Robots.” *Kinova*, www.kinovarobotics.com/product/gen3-robots.

[11] “Depth Camera D415.” *Intel® RealSense™ Depth and Tracking Cameras*, 10 Oct. 2022, www.intelrealsense.com/depth-camera-d415/.