ROS-BASED CONTROL OF A MANIPULATOR ARM FOR BALANCING A BALL ON A PLATE

Khasim Ali Khan, MS
Department of Mechanical Engineering
Northern Illinois University, 2017
Ji-Chul Ryu, Director

Automation and robotics are the growing phenomena replacing human labor in the industries. This idea of robots replacing humans is positively influencing the business thereby increasing its scope of research. This thesis discusses the development of an experimental platform to control a robotic arm through ROS (Robot Operating System) – open-source framework for robot software development providing advanced capabilities for several types of robots. In ROS, the major functioning is divided into several nodes that may receive or send messages to various platforms of sensors, states, and actuators. Furthermore, since ROS already has many written development packages, it avoids dealing with an opaque vendor API (application programming interface) or writing new device drivers in robot programming.

The robotic experimental platform developed in this thesis consists of a 7-DOF manipulator arm (Robai Cyton Gamma 300) equipped with a gripper, a vision tracking system with a low-cost web camera (PlayStation Eye), and a plate held by a gripper on which a ball is balanced. The robot arm is configured to create the plate motion in two perpendicular axes by using two joint actuators and this motion is further modeled into two decoupled, linearized systems using approximation linearization around the equilibrium point, the center of the plate. For demonstration purpose, a simple PD (proportional-derivative) control is used with the
positions of the plate tracked by the actuator encoders and of the ball by the vision system. To implement the controller, a joint velocity controller for the actuators (Robotis Dynamixel servo motors) is created in ROS and OpenCV vision libraries are used to write a control program code in C++. In this thesis, taking advantage of the interplatform operability of ROS, a system interface for a robot control is developed in which tracking an object, operating various actuators, and having a low-level control are possible with ease of programming. The system could perform some form of balancing act, but it needs further improvement. As future work, this system could be integrated with a mobile base to form a mobile manipulator for wider applications.
ROS-BASED CONTROL OF A MANIPULATOR ARM FOR BALANCING
A BALL ON A PLATE

BY

KHASIM ALI KHAN
© 2017 Khasim Ali Khan

A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

Thesis Director:
Ji-Chul Ryu
ACKNOWLEDGEMENTS

Alhamdulillah, this thesis would have not been possible without the support of many people who helped and supported me all throughout my graduation to make this journey one that I will cherish forever. I owe my gratitude to all those people who have made this thesis possible. Foremost, I would like to express my sincere gratitude to my advisor, Dr. Ji-Chul Ryu, for his continuous support all throughout my master's study and research, for his patience, motivation, and immense knowledge. His guidance and support helped me overcome many crisis situations and complete this thesis.

Besides my advisor, I would like to thank my thesis committee: Dr. Brianno D. Coller and Dr. Pradip Majumdar for their encouragement and insightful comments.

I thank my fellow mates Syed, Amin, Furqan, Sujith, Sandeep and Maxwell for all their effort, hard work, valuable comments and feedback.

Last but not the least, a very special thanks to very special people, my parents Tasneem Fatima and Vizarat Ali Khan and my brother Wajahat Ali Khan, for always supporting me throughout my achievements.
DEDICATION

To my parents, brother and family with love
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
</tbody>
</table>

**Chapter**

1. **INTRODUCTION**
   1.1 Background and Motivation
   1.2 Objective
   1.3 Literature Review

2. **SYSTEM HARDWARE OVERVIEW**
   2.1 Robai Cyton Gamma 300
   2.1.2 Joint Actuators
   2.2 Ball-on-Plate system
   2.3 Vision Camera System

3. **SOFTWARE**
   3.1 Introduction to ROS
   3.2 Programming on ROS
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>21</td>
</tr>
<tr>
<td>4.</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>31</td>
</tr>
<tr>
<td>5.</td>
<td>32</td>
</tr>
<tr>
<td>5.1</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>35</td>
</tr>
<tr>
<td>6.</td>
<td>38</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>39</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>41</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>45</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1 Fully automated manufacturing of a BMW X3 ......................................................... 1
Figure 2 NimRo-OP robot hardware.......................................................................................... 4
Figure 3 Mitsubushi PA 10 with ROS-based depth data............................................................ 5
Figure 4 Hardware setup of KUKA robot for stereo vision....................................................... 8
Figure 5 Physical dimensions of Robai Cyton Gamma 300 in millimeters.............................. 10
Figure 6 Experimental Setup of ball-on-plate on the Robai Cyton and the vision system........ 13
Figure 7 PlayStation Eye .......................................................................................................... 14
Figure 8 ROS filesystem levels................................................................................................. 18
Figure 9 Flow of messages in ROS.......................................................................................... 18
Figure 10 Architecture of system on ROS................................................................................ 19
Figure 11 Switching a Dynamixel to wheel mode................................................................. 21
Figure 12 Tracking with the camera ......................................................................................... 23
Figure 13 Coordinate system of the ball on the plate ............................................................... 25
Figure 14 Movement of ball along Y-axis.................................................................................. 25
Figure 15 Movement of ball along X-axis................................................................................ 27
Figure 16. Schematic of the feedback control loop ................................................................. 30
Figure 17 Mass properties of the link from SolidWorks .......................................................... 33
Figure 18. Simulation results. (a) Rotational motion about Y-axis (see Fig. 14); (b) rotational motion about X-axis (see Fig. 15)................................................................................. 35
Figure 19. Actual ball positions ................................................................. 36

Figure 20. Target and actual joint angular velocities. (a) Joint about Y-axis; (b) joint about X-axis (see Figures 14 and 15 for the corresponding rotational motions, respectively) .................. 36

Figure 21 Balancing act with velocity as input .............................................. 37
LIST OF TABLES

Page

Table 1 Specification of Robai Cyton Gamma 300 ................................................................. 11
Table 2 Specifications of PlayStation Eye .................................................................................. 14
Table 3 System Parameter Values Used in Simulation and Experiment ................................. 32
1. INTRODUCTION

1.1 Background and Motivation

From the first stone axe to the latest assembly lines in production, technology has long played a significant role in human effort. Machines have now replaced employees in the industries. Many current production lines involve the use of robots or automated procedures where human labor is entirely replaced by machines, keeping them safe from dangerous assembly operations (Fig. 1). This automation is positive for business not only in terms of reducing human labor but also increasing work hours, production rates and profit margins.

Figure 1 Fully Automated manufacturing of a BMW X3 (Source www.bimmer-mag.com).
Robotics is the stepping stone towards automation. It is the interdependent combination of mechanical, control systems, electrical and computer science engineering during the design process. Robotics breaks the traditional engineering boundaries emerging as a young modern field. Modern manufacturing engineering applications have laid down various research fields in robotics to be applied in manufacturing. Robotics along with automation has not only entered the business and trade but also achieved greater success in the fields of medicine, space research, transportation, entertainment and various multitudes of life.

Controlling these robots is also a task and there are many approaches for it. In simple terms the functionality of the robots is described by their manufacturers either providing their own user interface (UI) or application programming interfaces (APIs) for further functionality. Using these interfaces provided by the manufacturers may only direct towards a limited set of applications. Using the APIs may be difficult, sometimes requiring intense knowledge. These inadequacies resulted in more research and development of the middleware packages. ROS is one such middleware package which has been claimed by the experts to be one of the best middleware packages with a vast community which keeps growing. This motivated me towards developing a platform on ROS which can work between various actuators and sensors to demonstrate a control system.

1.2 Objective

The goal of this thesis is to have a solid interface for a manipulator arm which could demonstrate an elaborate task of reading in values from sensors, planning control action and
controlling actuators all under one platform. ROS, which is an open-source middleware package working between multiple platforms, was chosen as the platform to achieve the objective of this thesis. The work is demonstrated through a task of balancing a ball on a plate controlled by a manipulator arm with a vision system, operating under a ROS-based software platform.

1.3 Literature Review

The following literature review provides an idea of the capabilities of ROS and the methods of integration of a robotic arm with a vision system.

Morgan Quigley et al. [1] gives an overview of ROS, stating that it facilitates with a layer of structured communication above the cluster of the host operating system which performs the computing. This paper discusses relating ROS to existing robot software frameworks along with an overview on the fundamental concepts, functionality and applications. It summarizes the goals of ROS as “peer to peer, tools-based, multilingual, thin, free and open-source.” The authors anticipate that ROS’s architecture can be used for various hardware platforms, runtime requirements and research settings.

Philip Allgeuer et al. [2] describe a software framework for NimbRo-OP, an open humanoid robot inspired by the Robot from Robotis – DARwin-OP (Fig. 2). This software framework is based on the Robot Operating System (ROS) middleware. In this paper, ROS provides functionality for hardware abstraction, visual perception, and behavior generation and is used to implement basic soccer skills. This robot can be a perfect example to show the multiplatform operability of ROS working with 20 actuators (Dynamixel motors), wide-angle camera and Wi-
Fi adapters (as shown in Fig. 2), all controlled with a Zotac Zbox nano XS PC. This robot has multitude of tasks it could perform, but this limits to only position control of Dynamixel motors.

Simon Puligny [3] introduced a method to control a PR2 robot using its Webots simulator interfaced with ROS framework. The Webots simulator can be used in various operating systems such as Windows, Mac OS X and Linux, which shows the broad spectrum of ROS’s
interoperability. ROS communication and programming for a complex system is explained and a tool to convert the robot’s URDF file into the simulated model inside Webots is also created.

Garcia, Gil, Llacer and Torres [4] discuss a robotic control scheme based on vision image-based visual servoing (IBVS) developed over ROS. This scheme uses only the visual information obtained from a camera to guide the robot to a desired pose. This paper presents a comparative study of the performance of depth-based IBVS estimating depth using three separate ways with a low-cost RGB-D sensor (Microsoft Kinect). The platform of ROS was used to process the visual servoing information and the communication to the PA-10 is over a network client. A detailed schematic of this communication between PA-10 and Kinect on ROS is shown in Fig. 3. This platform uses the Kinect to analyse the control over the position of robot arm about objects that are to be manipulated.

![Figure 3 Mitsubushi PA 10 with ROS-based depth data - extracted from [4].](image-url)
While relevant research on ROS is discussed in the previous paragraphs, the following are a few of the previous works related to solving the problem of ball balancing on a plate, which is chosen for this thesis to demonstrate integration of actuators and sensors under ROS.

The ball on plate can be viewed as an extension to the ball-on-beam problem. Bolivar-Vincenty and Beauchamp-Baez [5] derive the equations of motion of the ball-on-beam system through the two most common methods of Lagrangian and Newtonian, which are obviously proven to be identical. Using the equation of motion, nonlinear state-space equations are developed. The slipping of the ball on the beam is neglected and the ball is assumed to roll under the influence of gravity. Nonlinear equations are linearized about the equilibrium point and the transfer function for controller design is obtained. The authors state that the Lagrangian method is easier to follow and understand than the Newtonian method, especially for complex systems.

Marta Virseda [6] does an in-depth study for the ball-on-beam system and works with two experimental implementations with the mathematical modelling of the system using Modelica. The beam is actuated by a DC motor and the ball is tracked by a Fire 1 camera vision system. The ball coordinates are sent via a network connection to a Linux PC running the control implementation in one of the plugins of Matlab. Several experimental processes were proposed and corroborated to demonstrate control theories in real systems.

Ming-Tzu Ho et al. [7] presented the design, implementation and validation of real-time visual servoing tracking control for a ball and plate system. Their experimental setup consists of two DC motors generating two-axes movement of the plate and the position of the ball are obtained by a machine vision system of a CMOS image sensor working at 150 Hz. The
Lagrangian method was used to derive the dynamic model of the system for analyses and simulation. Higher order coupling terms were neglected to simplify the ball-on-plate system into two decoupled ball and beam systems by assuming small angle movement of plate. The approach of approximate input-output feedback linearization was adopted for designing a controller to do the trajectory tracking. The control system developed was evaluated through simulations and experiments.

Galvan-Colmenares and Moreno-Armendariz [8] present the dual PD control for ball and plate system. They consider a novel nonlinear ball-on-plate system as a continuation to the ball-on-beam problem. Stability analysis is performed to evaluate the results. Nonlinear equations are attained using the Lagrangian method and the ball is controlled by a dual PD regulation controller. Minimum resources were utilized for the experiment: DC motors for titling axes actuation and an overhead camera ball position capture. Lyapunov stability analysis is carried out to prove that the system is asymptotically stable. Experimental results validate the feasibility of dual PD controller. As a matter of fact, the system shows a decoupled behavior for small angles at slow speeds, but adding the compensator makes the system more stable for a larger range of tilt.

Nathaniel Wetts [9] presents the dynamics of a ball-on-plate system mounted on a robot arm. The nonlinear system is obtained by the Lagrangian method and then linearized to design an LQR controller. The hardware setup includes a 7-DOF KUKA robot controlled by its own KRC controller, two color cameras which provide a stereo vision that are processed by ROS as shown in Fig. 4. A Kalman filter is introduced to estimate the state of the camera.
measurements. The experiment was conducted on a basketball and a gymnastic ball with the latter doing better because of the slight errors caused by the grooves of the basketball.

Figure 4 Hardware setup of KUKA robot for stereo vision - extracted from [9].
2 SYSTEM HARDWARE OVERVIEW

The experimental setup developed in this thesis consists of a 7-DOF robotic arm, Robai Cyton Gamma 300, having seven Dynamixel (DC servo) motors on each joint integrated with the vision system of a low-cost USB camera (PlayStation Eye). The actuators and vision components have their respective packages created in ROS. In this chapter, I explain the system from the hardware perspective. The control communication of these components on ROS and the system modelling and control are explained in the following chapters.

2.1 Robai Cyton Gamma 300

Robai Cyton Gamma 300, the manipulator arm chosen for the demonstration, has the same configuration as its industrial counterparts. This robot shows fluidic motion with seven degrees of freedom, making it easier to move around freely, acting like a human arm. It can easily perform the actions of picking and placing objects. Combined with its control software provided from the manufacturer, advanced control operations and visualizations can be performed by exploiting its kinematic redundancy. The physical dimensions of the Robai Cyton Gamma 300 are shown in Fig. 5 which are necessary for system modelling in Chapter 4. The detailed specifications of the robot are provided in Table 1.
Figure 5 Physical dimensions of Robai Cyton Gamma 300 in millimeters (Source www.robai.com).
### Table 1 Specifications of Robai Cyton Gamma 300

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Payload:</td>
<td>300 g (10.6 oz) at full extension</td>
</tr>
<tr>
<td>Total Reach:</td>
<td>53.4 cm (21.4 in)</td>
</tr>
<tr>
<td>Arm Weight:</td>
<td>1.2 kg (2.6 lbs)</td>
</tr>
<tr>
<td>Maximum linear arm speed:</td>
<td>20 cm/sec</td>
</tr>
<tr>
<td>Maximum joint speed:</td>
<td>30 rpm</td>
</tr>
<tr>
<td>Repeatability:</td>
<td>+/-1 mm</td>
</tr>
<tr>
<td>Gripper opening:</td>
<td>3.5 cm</td>
</tr>
<tr>
<td>Axes/DOF:</td>
<td>7</td>
</tr>
<tr>
<td>Shoulder Roll:</td>
<td>300°</td>
</tr>
<tr>
<td>Shoulder Pitch:</td>
<td>210°</td>
</tr>
<tr>
<td>Elbow Roll:</td>
<td>300°</td>
</tr>
<tr>
<td>Elbow Pitch:</td>
<td>210°</td>
</tr>
<tr>
<td>Wrist Yaw:</td>
<td>210°</td>
</tr>
<tr>
<td>Wrist Pitch:</td>
<td>210°</td>
</tr>
<tr>
<td>Wrist Roll:</td>
<td>300°</td>
</tr>
<tr>
<td>Interface:</td>
<td>Serial or USB</td>
</tr>
<tr>
<td>Software:</td>
<td>Actin library and flexible GUI/simulator, with inverse kinematics</td>
</tr>
</tbody>
</table>
2.1.2 Joint Actuators

The manipulator arm has Dynamixel DC servo motors as joint actuators. The motors are daisy chained for communication. As shown in Table 1, various links have different operating physical limits required during the system modelling. The low-level position control of motor and the sensor management is carried out by the microcontroller integrated in each servo motor. The motors can handle a voltage of up to 18 V. There is a two-wire cable going to the ‘USB2DYNAMIXEL’ converter which is required for communication between the computer and Dynamixel motors.

2.2 Ball-on-Plate system

A plate is held by a gripper-type end-effector of the robot arm and the camera is mounted to capture entire area of the plate as shown in Fig. 6. Various materials such as plastic, cardboard, glass and foam were evaluated for the plate. One crucial factor to consider is that the maximum payload of the robot is 300 grams at its stretched position. The plate is given two-axes motion by using two joints of the robot, which is explained in more detail in Chapter 4. After testing of the various materials, foam and soft rubber were selected for the plate and the ball, respectively, considering the payload limitation. Although the model-based control algorithm to be proposed in this thesis is supposed to work with any mass and size of the ball in principle, a smaller ball was considered to provide enough workspace on the plate along with the payload constraint.
2.3 Vision Camera System

For the vision system, a low-cost camera (Sony PlayStation Eye shown in Fig. 7) was considered. The camera can be used to capture image frames at a rate of up to 120 Hz with a resolution of 320×240 pixels. Although there is no official support or drivers from the manufacturer to run the camera, community-supported drivers are available for Mac OS and Linux. For Windows OS, there is a commercially available driver that can make it run like a regular webcam. A summary of the specifications of this camera are shown in Table 2.
Figure 7 PlayStation Eye (Source www.amazon.com).

Table 2 Specifications of PlayStation Eye

<table>
<thead>
<tr>
<th></th>
<th>Gaming Webcam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>640×480 at 60 Hz</td>
</tr>
<tr>
<td></td>
<td>320×240 at 120 Hz</td>
</tr>
<tr>
<td>Connectivity</td>
<td>USB 2.0</td>
</tr>
<tr>
<td>Platforms</td>
<td>Windows, Mac OS, Linux</td>
</tr>
</tbody>
</table>
3. SOFTWARE

Robai Cyton Gamma comes with its own software – the CytonViewer, which has more of the simulation capabilities. There is also a Labview interface which uses pieces of drivers provided by the manufacturer with limited functionality. We needed a more robust system which gives elaborate control over the robot and integrate it with the sensors for feedback. Hence, the approach to use a middleware package–ROS– was considered to get a working control system. In ROS, we directly control the Dynamixel motors of the Cyton hardware which gives us the ability to explore the complete functionality of the robot. A detailed description of ROS along with its functionality is provided in this chapter.

3.1 Introduction to ROS

ROS, developed in Stanford Artificial Intelligence Laboratory (SAIL) in 2007, is an open-source framework distributed under BSD (Berkeley Software Distribution) license and integrated development environment for developing robot software to create complex and sound robotic functionality. It is a middleware software distribution and an operating system like Windows and Linux based on UNIX architecture having a set of libraries that are perfect to build a new type of robot structure on a heterogeneous cluster. It comes equipped with visualization,
simulation, and debugging tools. ROS allows the users to have distributed operations over clusters, multiprocessor environments and GPUs in multicore. [10]

The idea behind ROS is to create modules of software that would also work in other robots with slight modification of a base code. ROS libraries under various open-source software licenses are responsible for implementing common functionality and applications of planning, hardware drivers, simultaneous localization and mapping (SLAM), robot models, datatypes, perception, simulation tools, and other algorithms. The overall intent is creating functionalities that can be shared and used in other robots without writing the whole code when it is already available in the libraries. This is important for this thesis since the libraries – the ROS packages for Dynamixel motors – are already developed and can be used to have control over our Robai Cyton arm.

ROS allows interplatform operability, packet management, low-level device control, code reusability, messaging between processes and distributed computing and can support various levels and types of coding languages. ROS is very useful for elaborate systems especially in multithreaded environments having features of concurrent resource handling. These features of ROS are suitable for our application where the robotic manipulator is integrated with a vision type feedback system [11]. A package of the vision system continuously tracks the position of the ball and communicates with the package for Dynamixel motors to perform the manipulation action.
3.2 Programming on ROS

ROS has been designed to work with Ubuntu Linux, Windows and Mac OS; while the platforms for Windows and Mac OS are “experimental”, they have limited functionality. Only Ubuntu Linux is considered supported. Following the durable architecture of UNIX, small generic programs are used to create complex software systems. The uniqueness of ROS lies in the fact that it does not have an integrated development and runtime environment. Architecture of ROS is “tools based” with separate programs for the tasks of data streaming, navigating the source code tree, visualizing the system connections, documentation, etc. [11]

Filesystem of ROS is divided into various small programs as shown in Fig. 8. A stack is a collection of various packages of similar functionality with a stack manifest giving information of dependencies and licensing of these packages. Most packages for the respective components of this thesis are available due to the huge ROS community. A package is a directory containing nodes (executables), external libraries, data, configuration files and one xml configuration file named manifest.xml. Each package created has its own messages and services which can be used to send information to other processes. The data types of these messages must be known.
Nodes, which are executables, perform the computation in a package. The communication between nodes as shown in Fig. 9 is with topics and services. A stream of messages of a certain type is a topic which can be used with multiple nodes, whereas a service is a one-to-one interaction to make a request from a node.
Packages for the Dynamixel motors of the robot arm and the vision system were available due to the huge community of ROS. Respective packages were required to be built and nodes were created. The master is a node declaration and registration service, that makes it possible for nodes to find each other and exchange data. Running these nodes can be through commands like ‘roslaunch’ or ‘rosrun’ with the roslaunch being typically used to run the master and the controllers and the rosrun for the system components (robot, vision system). The main code to take the inputs from the system and give commands to the actuators is also written as a node – motion planning node. The structure of the ROS-based software system proposed in this theses is shown in Fig. 10 taking inputs and performing the control action.

![Figure 10 Architecture of system on ROS](image)

3.3 ROS with Robai Cyton Gamma 300

The manufacturer of Robai Cyton Gamma provides a very limited support for ROS, but a few working systems with ROS have been reported. In his pioneering work on Robai Cyton with
ROS, Andres Fekete [12] modified the ROS packages to integrate the Cyton hardware directly accessing the Dynamixels without using the software provided by Robai. Inspired by this work, the approach of low-level control of Dynamixel motors by creating the ROS packages was adopted. In the implementation of the suggested approach, it is important to understand two control modes of Dynamixel motors: joint position control and joint torque control modes. In the joint torque control mode, the input is in the form of velocity and not the torque. When the Dynamixel motors work as a part of the original Cyton hardware, they run in the joint position control mode with their position limits set in the configuration file based on the hardware configuration. However, this thesis is aimed to give the input in the form of velocity. Therefore, the respective velocity controllers for the Dynamixels were created in ROS and the default joint control mode was necessary to change to wheel mode (velocity control mode). In the wheel mode, the clockwise and anti-clockwise limits of the Dynamixel were eliminated to acquire free continuous spin movement. The images in Fig. 11 show the process of changing the motor configuration using the Dynamixel wizard.
A joint velocity controller was defined by first creating a configuration file (.yaml) in which all the necessary parameters were introduced. Then a launch file was created to load all the controller parameters to the server and start up the controller. Controller was then loaded to send velocity commands that run the Dynamixel motors. It is very important to define the working limits of the Dynamixels since, in the wheel mode, free spin movement will hit the physical boundaries of the robot. This was done by giving the input of zero velocity when the encoders read joint position values beyond the working range of our application.

3.4 ROS with Vision Camera System

A vision system is required to track the position of the ball on the plate. Image processing is conducted to track a certain colored object (in our case, a red ball) using the open CV libraries.
The ROS packages for vision systems are used and the tracking algorithm is written as another package. The information from these tracking packages is used to track the ball position which is used as input in the task of ball balancing on the plate.

The series of image processing for tracking is as follows:

- Video stream from the camera is received
- Image is converted to grey scale
- Image is converted to HSV scale
- HSV value is set to detect the red object
- Moments of the red objects calculated using “moments” of open CV
- The centroid of the red objects is calculated and stored
- This value is published in the ROS node

This vision camera system can capture image frames up to 120 Hz at a resolution of 320×240 pixels. Hence a window of 320×240 pixels in the vision program is created to cover the area of 320×240 mm² such that 1 pixel would represent 1 mm of distance. Then the camera is mounted in its fixture above the plate (see Fig. 6) at a height that allows it to capture the complete plate area of 320×240 mm². The frame rate was set at 100 Hz in the experiment conducted in Chapter 6. Figure 12 shows the screenshot of the vision tracking module in operation. The code for the tracking is given in Appendix A.
Figure 12 Tracking with the camera
4. SYSTEM MODELLING AND CONTROL

The experimental setup is shown in Fig. 6 of the Chapter 2. The plate is held by the gripper of the robot and the camera is placed over the plate on a stand capturing the entire area of the plate. The zero position of the two motors, which give two-axes motion to the plate, is set such that the plate stays horizontal. The operating position limits of the Dynamixel motors are set to prevent them from physically interfering with the other links of the robot arm. For control purpose, the entire ball-on-plate system on the robot arm is simplified into two decoupled ball-on-beam systems and the motion of the ball in each axis is treated as an independent ball-on-beam problem. The ball is assumed to be in contact with the plate all the time and the ball rolls without slipping.

4.1 Equations of Motions

The Lagrangian method is used to derive the equations of motion. The coordinate system of the ball on plate is shown in Fig. 13.
For the motion of the plate about Y-axis, the system can be considered a ball-on-beam system as shown in Fig. 14. The parameters of the ball are given by mass $m_b$, radius $r_b$, moment of inertia $J_b$ and the plate by moment of inertia $J_p$ about Y-axis.

Using the body-attached coordinate frame $u_1$–$u_2$ as defined in Fig. 14, the position of the ball is then written as

$$x = x_b \hat{u}_1$$

By differentiating the position vector, the velocity of the ball is given by

$$\mathbf{v}_b = \frac{dx}{dt} = \dot{x}_b \hat{u}_1 + x_b \dot{\theta}_y \hat{u}_2$$
resulting in $v_b^2 = \|v_b\|^2 = \dot{x}_b^2 + (x_b\dot{\theta}_y)^2$. With the no slip assumption that $\dot{x}_b = r_b\omega_y$, where $\omega_y$ denotes the angular velocity of the ball about the direction perpendicular to the page, we have

$$\omega_y = \frac{\dot{x}_b}{r_b}$$

With the above expressions, the kinetic and potential energies of the system are given by

$$K = \frac{1}{2}m_b v_b^2 + \frac{1}{2}I_b \omega_y^2 + \frac{1}{2}J_p \dot{\theta}_y^2$$

$$= \frac{1}{2}\left( m_b + \frac{I_b}{r_b^2} \right) \dot{x}_b^2 + \frac{1}{2} \left( m_b x_b^2 + J_p \right) \dot{\theta}_y^2$$

$$V = m_b g x_b \sin \theta_y$$

Using the Euler Lagrange equation with $q = [x_b, \theta_y]^T$:

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}} \right) - \frac{\partial L}{\partial q} = 0$$

where $L = K - V$, we get the equations of motion as follows:

$$\left( m_b + \frac{I_b}{r_b^2} \right) \ddot{x}_b - m_b x_b \dot{\theta}_y^2 + m_b g \sin \theta_y = 0 \quad (1)$$

$$\left( m_b x_b^2 + J_p \right) \ddot{\theta}_y + 2 m_b x_b \dot{x}_b \dot{\theta}_y + m_b g x_b \cos \theta_y = \tau_y \quad (2)$$

where $\tau_y$ denotes the torque input on the Y-axis joint.

The motion of the plate about X-axis can also be considered a ball-on-beam system, but in this case with its desired equilibrium position off from the axis of rotation ($y_b \neq 0$). In addition, the center of mass is not at the axis of rotation as shown in Fig. 15; this results in having additional potential and kinetic energies.
The links with the gripper and plate (as shown in Fig. 15) are as a whole considered as the system which contributes to the additional potential and kinetic energies. The total kinetic and potential energies are now given as follows:

\[ K = \frac{1}{2} m_b v_b^2 + \frac{1}{2} J_b \left( \frac{\dot{y}_b}{r_b} \right)^2 + \frac{1}{2} m_s v_p^2 + \frac{1}{2} J_s \dot{\theta}_x^2 \]

\[ \mathcal{V} = m_b g y_b \sin \theta_x + m_s g l_c \sin \theta_x \]

where \( m_s \) and \( J_s \) are the mass and moment of inertia of the entire system including the links, gripper, and plate; \( l_c \) is the location of the COM measured from the axis of rotation; and \( v_p \) is the velocity of the system at the COM, given by \( v_p = l_c \dot{\theta}_x \).

Hence the equations of motion with the Lagrangian are obtained as follows:

\[ \left( m_b + \frac{J_b}{r_b^2} \right) \ddot{y}_b - m_b y_b \dot{\theta}_x^2 + m_b g \sin \theta_x = 0 \quad (3) \]

\[ (m_b y_b^2 + J_s + m_s l_c^2) \ddot{\theta}_x + 2 m_b y_b \dot{y}_b \dot{\theta}_x + (m_b y_b + m_s l_c) g \cos \theta_x = \tau_x \quad (4) \]
4.2 Linearization

The ball is targeted to be balanced at the center of the plate and the approximate linearization with assumptions of small angles of rotation make it simpler. The linearization is done as follows for the equations (1) through (4);

(i) For the decoupled system along the Y-axis, equations (1) and (2) can be represented in the state-space form as follows:

\[
\dot{Z} = \begin{bmatrix}
\dot{x}_b \\
\dot{\theta}_y \\
\ddot{x}_b \\
\ddot{\theta}_y
\end{bmatrix} = \begin{bmatrix}
\dot{x}_b \\
\dot{\theta}_y \\
-\frac{5}{7} g \sin \theta_y + \frac{5}{7} x_b \dot{\theta}_y^2 \\
\frac{1}{m_b x_b^2 + f_p} (-2 m_b x_b \dot{x}_b \dot{\theta}_y - m_b g x_b \cos \theta_y + \tau_y)
\end{bmatrix}
\]

The system is linearized about the equilibrium point \( Z^\ast \) where \( x^*_b, \theta^*_y, \dot{x}^*_b, \dot{\theta}^*_y \) and \( \tau^*_y \) are all set to be 0.

Linearization of the system is done as follows:

\[
\dot{Z} = f(Z, \tau)
\]

\[
\frac{\partial f_i}{\partial Z_j} \text{ at equilibrium} = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & -\frac{5}{7} g & 0 & 0 \\
-\frac{m_b g}{f_p} & 0 & 0 & 0
\end{bmatrix} = A
\]
\[ \frac{\partial f_i}{\partial \tau} \text{ at equilibrium} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ J_p \end{bmatrix} = B \]

Hence the linearized equations for the Y axis are as follows:

\[ \ddot{x}_b + \frac{5}{7} g \dot{\theta}_y = 0 \]  
\[ J_p \ddot{\theta}_y + m_b g x_b = \tau_y \]

(ii) For the decoupled system about the X-axis, equations in the state-space form are written as

\[ \dot{Z} = \begin{bmatrix} \dot{y}_b \\ \dot{\theta}_x \\ \dot{y}_b \\ \dot{\theta}_x \end{bmatrix} = \begin{bmatrix} \dot{y}_b \\ \dot{\theta}_x \\ -\frac{5}{7} g \sin \theta_x + \frac{5}{7} y_b \dot{\theta}_x^2 \\ \frac{1}{m_b y_b^2 + J_s + m_s l_c} \left(-2m_b y_b \dot{y}_b \dot{\theta}_x - (m_b y_b + m_s l_c) \cos \theta_x + \tau_x \right) \end{bmatrix} \]

The system is linearized about the equilibrium point where \( \dot{y}_b^*, \dot{\theta}_x^*, \dot{\theta}_x^* \) and \( \tau_x^* \) are set to be 0. But the equilibrium point of \( y_b^* \) is not zero, since the equilibrium point is off the axis of rotation by \( y_b^* \) as seen in Figure 16. Therefore, the equilibrium torque \( \tau_x^* \) is not zero either.

\[ y_b^* \neq 0 \]

\[ \tau_x^* = (m_b y_b^* + m_s l_c) g \]
Due to the nonzero equilibrium point, the approximation linearization yields

$$\dot{\Delta z} = A\Delta z + B\Delta \tau_x$$  \hspace{1cm} (8)

where

$$\Delta z = z - z^*$$
$$\Delta \tau_x = \tau_x - \tau_x^*$$

$$A = \left. \frac{\partial f}{\partial z} \right|_{z^*,\tau_x^*} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{5}{7}g & 0 & 0 \\ -\frac{m_b g}{m_b y_b^* + J_s + m_s l_c^2} & 0 & 0 & 0 \end{bmatrix}$$

$$B = \left. \frac{\partial f}{\partial \tau_x} \right|_{z^*,\tau_x^*} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

And hence, the linearized equations about the X axis are

$$\ddot{y}_b + \frac{5}{7}g \theta_x = 0$$  \hspace{1cm} (9)

$$(m_b y_b^* + J_s + m_s l_c^2) \ddot{\theta}_x + m_b g (y_b - y_b^*) = \tau_x$$  \hspace{1cm} (10)
4.3 Controller Design

The controller of the system is individually designed for each axis, represented by equations (5), (6) and (9), (10), respectively. The technique of pole placement is used to design a PD-like controller in the form of

\[ \tau = -Kz \]  \hspace{1cm} (11)

where \( \tau \) represents the torque control input, \( z \) is the full state feedback, and \( K \) is the gain matrix.

The system equation is then obtained as

\[ \dot{z} = (A - BK)z \] \hspace{1cm} (12)

The stabilization to the equilibrium point is guaranteed by determining the gain matrix \( K \) in a way that \( (A - BK) \) is asymptotically stable – Hurwitz matrix.

It should be mentioned that the Cyton Gamma manipulator arm has limited input functionality. The robot does not allow torque input to the system but velocity input. To overcome this limitation, the approach considered was to numerically integrate the torque and predict the velocity at the next control step. The state-space equations were used to get the acceleration of the links, which was then integrated numerically to predict the velocity required at the next step and given as control input to the system. Giving velocity control input to Robai Cyton is explained in Chapter 3. The schematic of the system with the controller is shown in Fig. 16. The complete C++ code for balancing using velocity as control input is given in Appendix B.
5 SIMULATION AND EXPERIMENTAL RESULTS

The modelled system equations and the proposed control technique have been evaluated in simulations done on Matlab. The validity of the system equations is checked by using the standard solver ode45 in Matlab. It should be noted that the simulations are based on the decoupled linearized equations which may not describe the exact motion of the ball-on-plate system but would closely depict the actual motion around the equilibrium (balancing) point. The system parameters used in simulations and experiments are listed in Table 3.

<table>
<thead>
<tr>
<th>System Parameters</th>
<th>Variables</th>
<th>Actual Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass of the ball</td>
<td>$m_b$</td>
<td>0.0045 [kg]</td>
</tr>
<tr>
<td>Moment of inertia of the ball</td>
<td>$J_b$</td>
<td>$1.046 \times 10^{-6}$ [kg⋅m$^2$]</td>
</tr>
<tr>
<td>Radius of the ball</td>
<td>$r_b$</td>
<td>0.024 [m]</td>
</tr>
<tr>
<td>Mass of the plate</td>
<td>$m_s$</td>
<td>0.0228 [kg]</td>
</tr>
<tr>
<td>Moment of inertia of the plate (about Y-axis)</td>
<td>$J_p$</td>
<td>$1.702 \times 10^{-6}$ [kg⋅m$^2$]</td>
</tr>
<tr>
<td>Moment of inertia of the plate (about X-axis)</td>
<td>$J_s$</td>
<td>$1.60577 \times 10^{-3}$ [kg⋅m$^2$]</td>
</tr>
<tr>
<td>Distance to the plate center from the rotation of axis</td>
<td>$y_b^*$</td>
<td>0.23 [m]</td>
</tr>
</tbody>
</table>
Since the rotating links which are considered part of the plate are irregularly shaped, the moment of inertia of the plate (together with the links) and the COM (Fig. 14) was determined using a model developed on SolidWorks as presented in Fig. 17. The mass of the system components (links, gripper) was measured on a scale and then used in SolidWorks to evaluate the moment of inertia and the center of mass.

![Mass properties of System2](image)

**Figure 17** Mass properties of the link from SolidWorks
5.1 Simulation Results

The pole placement method was used to obtain the gain matrix $K$. The poles were chosen such that the system has a small overshoot since the operating area of the plate was small. This gain matrix $K$ was then used to obtain the controlled system equation given by Eq. (12). The simulation was conducted with the initial conditions $\left( x_b, \theta_y, \dot{x}_b, \dot{\theta}_y \right) = (0.1 \text{ m}, 0, 0, 0)$ and $\left( y_b, \theta_x, \dot{y}_b, \dot{\theta}_x \right) = (0.33 \text{ m}, 0, 0, 0)$ for motions about Y-axis and X-axis respectively. Note that the initial condition of $y_b = 0.33 \text{ m}$ indicates that the ball is actually 0.1m away from the desired equilibrium position, i.e., the center of the plate. (See Chapter 4 for the origin of the coordinate frame).

The simulation results in Fig. 18 (a) and (b) show that the system reaches the equilibrium position at about $t = 3$ seconds. The graph on Fig. 18 (b) shows that the position of the ball $y_b$ reaches 0.23 m which is the desired equilibrium position on the X-axis. It is observed that the simulation yields precise results and all the states reach the equilibrium point. We do not present them here, but similar successful simulation results were achieved for different values of the ball and plate parameters when using the approximation linearization and the simplification of considering two independent ball-on-beam problems.
5.2 Experimental Results

The demonstration setup was able to perform some form of balancing act, but not satisfactorily to balance the ball at the center of the plate. The measured position of the ball during the experiment is shown in Fig. 19. The variations between the desired velocities and actual velocities for the Y-axis and X-axis are shown in Fig. 20 (a) and (b), respectively.
Figure 19. Actual ball positions

Figure 20. Target and actual joint angular velocities. (a) Joint about Y-axis; (b) joint about X-axis (see Figures 14 and 15 for the corresponding rotational motions, respectively)

The response of the actuators (Dynamixel motors) was not fast enough to keep the ball on the surface of the plate while the plate movement was in the proper direction. The speed of movement of the plate was based on deviation of ball position from the center of the plate. The
plate was added with borders (as seen in Fig. 21) to prevent the ball from falling off the plate due to unsuccessful control performance, in order to show the control action and obtain the experimental data. The approach of considering two decoupled ball-on-beam systems, small angle approximation and giving the velocity control (instead of the torque) appears to dilute the performance of the system. Other factors could be the delays in image processing and communication and slower control loop rate. Also, the velocity controller in the Dynamixel motors uses PI control internally to reach the goal velocity, so there might also be a time lag for the Dynamixel motor to reach the specified velocity (given as input). Other linear control techniques such as LQR to optimize the control action could be further applied to improve the task performance.

![Figure 21 Balancing act with velocity as input](image)
6 CONCLUSION

In this thesis, a solid experimental platform to control a robotic arm through ROS was developed, which could lead to further robot software development providing advanced capabilities for various types of robots. The 7-DOF manipulator arm – Robai Cyton Gamma 300 – equipped with a gripper is integrated with a vision tracking system (PlayStation Eye) to demonstrate the task of ball balancing on a plate. The system is designed to operate by receiving feedback from the vision system and states of the motors, compute the control input using the proposed algorithm, and sending the input commands to the actuators. The motion of the ball on plate is modeled as two decoupled, linearized systems using approximate linearization about the equilibrium point. A simple PD controller is used to develop the control action for the actuators of Dynamixel motors. All these processes take place under the developed ROS platform taking advantage of its interplatform operability. As a future work, the balancing act on the system is to be refined. This platform is to be further integrated with a mobile base (Pioneer D3X) forming a mobile manipulator to perform more complex tasks.
REFERENCES


[A] C++ code for tracking a red colored object

```cpp
#include<ros/ros.h>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <geometry_msgs/Point.h>

using namespace cv;
using namespace std;

// Values for Red color ball
int thresh1 = 0;
int thresh2 = 15;
int sval1 = 85;
int sval2 = 255;
int vval1 = 40;
int vval2 = 255;

int main(int argc,char** argv) {  
    ros::init(argc, argv, "camera_tracker");
    ros::NodeHandle n;
    ros::Publisher pub = n.advertise<geometry_msgs::Point>("point", 100);
    ros::Rate rate(100);
    VideoCapture stream1(1);  // 0 is the id of video device. 0 if you have only one camera.
    stream1.set(CAP_PROP_FRAME_WIDTH, 320);
    stream1.set(CAP_PROP_FRAME_HEIGHT, 240);
```
stream1.set(CAP_PROP_FPS, 100);

if (!stream1.isOpened()) { //check if video device has been initialised
    cout << "cannot open camera";
}

namedWindow("ctl", 1);
createTrackbar("Hue", "ctl", &thresh1, 255);
createTrackbar("Hue2", "ctl", &thresh2, 255);
createTrackbar("Sat1", "ctl", &sval1, 255);
createTrackbar("Sat2", "ctl", &sval2, 255);
createTrackbar("Val1", "ctl", &vval1, 255);
createTrackbar("Val2", "ctl", &vval2, 255);

while(ros::ok())
{
    Mat src, src_hsv, src_gray, dst, dst2;
    stream1.read(src);
    cvtColor(src, src_gray, cv::COLOR_BGR2GRAY);
    cvtColor(src, src_hsv, cv::COLOR_BGR2HSV);
inRange(src_hsv, Scalar(thresh1, sval1, vval1), Scalar(thresh2, sval2, vval2), dst);
    cv::Moments m = moments(dst, true);
    double area = countNonZero(dst);
    double m10 = m.m10;
    double m01 = m.m01;
    double m00 = m.m00;
    int posx = m10 / m00;
    int posy = m01 / m00;
    if (area < 15 * 15) { posx = 160; posy = 120; }
cvtColor(dst, dst2, cv::COLOR_GRAY2BGR);
circle(dst2, Point(posx, posy), 10, Scalar(0, 0, 255), -1);

imshow("cam", src);
imshow("fil", dst2);
if (waitKey(30) >= 0)
    break;

geometry_msgs::Point msg;
msg.x = posx;
msg.y = posy;
pub.publish(msg);
ros::spinOnce();
rate.sleep();
}
return 0;
}
APPENDIX B
[B] C++ code for moving the robot with velocity controller.

```cpp
#include<ros/ros.h>
#include<std_msgs/Float64.h>
#include <geometry_msgs/Point.h>
#include "dynamixel_msgs/JointState.h"
#include "dynamixel_msgs/MotorStateList.h"
#include<stdio.h>
#include <stdlib.h>
#include <math.h>
#include <fcntl.h>
#include <sys/time.h>
#include <time.h>
#include <fstream>

double getcurrtime() {
    struct timeval tv;
    gettimeofday(&tv,NULL);
    printf("Time: %d %d\n", tv.tv_sec, tv.tv_usec);
    return (double)tv.tv_sec + (double)tv.tv_usec / 1.0e6;
}

using namespace std;

class Ball {
    public:
        static Ball *_singleton;

```
ros::Subscriber sub;

double px, py, vx, vy, currtime;

double prevx, prevy, prevtime;

Ball() {
   _singleton = this;
   px = py = 0;
}

void subscribe(ros::NodeHandle& n) {
   sub = n.subscribe("/point", 1, pointCallback);
}

static void pointCallback(const geometry_msgs::Point::ConstPtr& msg)
{
   float x = float(msg->x - 160) / 160;
   float y = float(msg->y - 120) / 120;

   std::cout << "Point: " << msg->x << ", " << msg->y << std::endl;
   Ball &This = *_singleton;
   This.prevx = This.px;
   This.prevy = This.py;
   This.prevtime = This.curtime;
   This.px = x*160/1000;
   This.py = (230-(-y*120))/1000;
   This.curtime = getcurrtime();
   printf("Cam time: %f\n", This.curtime);

   float t = This.curtime - This.prevtime;
   if (t == 0) return;
double alpha = 0.4;

This.vx = This.vx * alpha + ((This.px - This.prevx) / t) * (1 - alpha);
This.vy = This.vy * alpha + ((This.py - This.prevy) / t) * (1 - alpha);

};

} ball;

Ball *Ball::_singleton = NULL;

class RobotInterface
{

public:

  struct Joint {
    double min, max, mean;
    double velocity;
    double position;
    double currTime;
    double prevPosition;
    double prevTime;
    const char *name;

    Joint() {
      velocity = 0;
      position = 0;
      currTime = 0;
      prevPosition = 0;
      prevTime = 0;
    }
  };
static RobotInterface * _singleton;

Joint joint_pitch, joint_roll;

ros::Subscriber sub1, sub2;
std::ofstream fout;

RobotInterface() {
  _singleton = this;
  joint_pitch.name = "joint_pitch";
  joint_pitch.min = -0.19;
  joint_pitch.mean = 1.54;
  joint_pitch.max = 0.26;
  joint_roll.name = "joint_roll";
  joint_roll.min = -0.65;
  joint_roll.mean = 3.13;
  joint_roll.max = 0.39;
  fout.open("state.log");
}

void subscribeState(ros::NodeHandle& n) {
  sub1 = n.subscribe("/pan_controller1/state", 1, stateCallback1);
  sub2 = n.subscribe("/pan_controller2/state", 1, stateCallback2);
}

static void stateCallback(const dynamixel_msgs::JointState::ConstPtr& msg, Joint& joint) {
  long s = (long)msg->header.stamp.sec;
  long ns = (long)msg->header.stamp.nsec;
  double t = s + (double)ns / 1000000000.0;

_singleton->fout << msg->header.stamp.sec <<
    "t" << msg->header.stamp.nsec <<
    "t" << joint.name <<
    "t" << msg->current_pos <<
    "t" << msg->velocity <<
    std::endl;

joint.prevPosition = joint.position;
joint.prevTime = joint.currTime;

double filter = 0.5;

joint.position = msg->current_pos - joint.mean;
joint.currTime = t;

joint.velocity = joint.velocity * filter + ((joint.position - joint.prevPosition) / (joint.currTime - joint.prevTime)) * (1 - filter);
}

static void stateCallback1(const dynamixel_msgs::JointState::ConstPtr& msg)
{
    stateCallback(msg, _singleton->joint_pitch);
}

static void stateCallback2(const dynamixel_msgs::JointState::ConstPtr& msg)
{
stateCallback(msg, _singleton->joint_roll);
}
} robot;

RobotInterface *RobotInterface::_singleton = NULL;

int main(int argc, char** argv)
{
    ros::init(argc, argv, "robot_balance");
    ros::NodeHandle n;
    ros::Publisher pub1 = n.advertise<std_msgs::Float64>("/pan_controller1/command", 1);
    ros::Publisher pub2 = n.advertise<std_msgs::Float64>("/pan_controller2/command", 1);
    ros::Rate r(100); // Hertz
    robot.subscribeState(n);
    ball.subscribe(n);

    int t = 0;
    double a1 = 0, a2 = 0, dt = 0;
    double vel1 = 0, vel2 = 0;
    while(ros::ok())
    {
        ros::spinOnce();

        printf("(%f, %f)\n", robot.joint_pitch.velocity, robot.joint_roll.velocity);
        printf("(%f, %f)\n", robot.joint_pitch.position, robot.joint_roll.position);
        printf("PointVel (%f, %f)\n", ball.vx, ball.vy);
        printf("PointPos (%f, %f)\n", ball.px, ball.py);
a1 = 29.4*ball.vy-15*robot.joint_pitch.velocity-83.8*robot.joint_pitch.position+27*ball.py-6.21;

a2 = 29*ball.vx-15*robot.joint_roll.velocity-84*robot.joint_roll.position+27*ball.px;

dt = 0.03;

if (robot.joint_pitch.position <= robot.joint_pitch.min || robot.joint_pitch.position >= robot.joint_pitch.max)
{
    vel1 = 0;
}
else
{
    vel1 = robot.joint_pitch.velocity + a1*dt;
}

if (robot.joint_roll.position <= robot.joint_roll.min || robot.joint_roll.position >= robot.joint_roll.max)
{
    vel2 = 0;
}
else
{
    vel2 = robot.joint_roll.velocity + a2*dt;
}

printf("FinalVel (%f, %f)\n", vel1, vel2);
printf("time (%f)\n", dt);
printf("acceleration (%f, %f)\n", a1, a2);

// Comment out the following to only display values and not publish to robot
std_msgs::Float64 msg1;
msg1.data = vel1;
pub1.publish(msg1);

std_msgs::Float64 msg2;
msg2.data = vel2;
pub2.publish(msg2);

r.sleep();
}