A

REPORT

ON

REAL TIME INTELLIGENT CONTROL SYSTEM FOR ROBOTIC MANIPULATOR THROUGH VISUAL SERVOING

BY

VRUSHALI B. PATIL 2016AATS0079U ECE

Submitted in partial fulfilment of

LABORATORY PROJECT COURSE ECE F366

UNDER GUIDANCE OF

DR. V. KALAICHELVI

BITS Pilani, Dubai Campus

Dubai International Academic City (DIAC)

Dubai, U.A.E

(JANUARY 2019 TO MAY 2019)

BITS PILANI, Dubai Campus

Dubai International Academic City (DIAC)

Dubai

A

REPORT

ON

REAL TIME INTELLIGENT CONTROL SYSTEM FOR ROBOTIC MANIPULATOR THROUGH VISUAL SERVOING

BY

VRUSHALI B. PATIL 2016AATS0079U ECE

Submitted in partial fulfilment of

LABORATORY PROJECT COURSE ECE F366

UNDER GUIDANCE OF

DR. V. KALAICHELVI

BITS Pilani, Dubai Campus

Dubai International Academic City (DIAC)

Dubai, U.A.E

(JANUARY 2019 TO MAY 2019)

BITS PILANI, Dubai Campus

Dubai International Academic City (DIAC)

Dubai

ABSTRACT

Duration: 23 Jan 19 to 11 May 19 Start Date: 23 Jan 19

Date of Submission: 11 May 19

Title: Real Time Intelligent Control System for Robotic Manipulator through Visual Servoing

Student Name: Vrushali Bhausaheb Patil **Student ID:** 2016AATS0079U

Discipline of Student: Electronics and Communication Engineering

Name of Faculty: Dr. V. Kalaichelvi

Keywords: Arduino, ROS, Motor Drivers, Timers

Abstract: The controller of the robot is same as a human brain and it controls the entire functioning of the robot, and also networks with other devices. Hence it becomes extremely important to have a full control over the controller in order to carry out important tasks using the manipulator. This report deals with the process of bypassing the existing TrioPC controller of the robot with a new custom designed controller. The new controller proves to be of great advantage specifically when: Custom trajectories are needed, inverse kinematics should be implemented, and when one has to control the robot using full ROS stack. A custom controller is designed using Arduino. Special codes to control individual motors have been written which send direction and pulse commands to the motor drivers. Feedback about joint limits is also obtained from homing sensors and used as a safety measure. The real time homing of the robot has been demonstrated along with a real time Visual Servoing application to justify the entire working of the controller.

Signature of Student Signature of Faculty in charge

Date: 11 May 2019

ACKNOWLEDGEMENTS

I owe the successful results in this work to a lot of people who have encouraged me to write the report. I would like to sincerely thank all those who are involved in this for their constant support and guidance throughout the project.

Firstly, thanks are due to Director Prof. Dr. RN Saha and Dr. Anand Kumar for providing the opportunity to take up this project. Next, I would like to express my deepest gratitude for my advisor, Dr. V. Kalaichelvi for readily agreeing to mentor and guide me through the project and motivating me at crucial moments. Thank you Ma'am for helping me stay committed to this endeavor. My design project work owes its significant value to you. I would also like to extend my sincere thanks to Dr. Ram Karthikeyan for valuable inputs and guidance.

I am eternally grateful to our librarian for the help in procuring necessary papers from sister campuses as well as from other sources.

I would like to also acknowledge the help of authors and website creators whose resources have helped me build this study project.

Finally, I would like to thank my parents and all my friends for their constant support and motivation and for dedicating time and effort towards this project.

Vrushali Patil

11 May 2019

CONTENTS

LIST OF FIGURES

LIST OFABBREVATIONS

DOF	Degrees of Freedom
ROS	Robot Operating System
PUL	PULSE
DIR	DIRECRTION
VS	VISUAL SERVOING

CHAPTER 1

BASIC THEORY

1.1 INTRODUCTION

Industry-specific robots perform many tasks such as grasping and placing objects at required place. Such robots are made to perform these tasks by observing the handing of manual tasks by a fully-functioning human arm. A robotic arms is also known as robotic manipulator. In early days these robotic arms were designed to replace humans while working with bio-hazardous or radioactive materials or for use in inaccessible places.

A series of sliding or joined segments are put together to form an arm-like manipulator that is capable of automatically moving objects within a given number of degrees of freedom. Every industrial robotic manipulator has a controller and an arm. Many factors determine the performance of the manipulator like its speed, payload weight and precision. However, the reach of its end-effectors, the overall working space and the orientation of the work is determined by the structure of the manipulator.

A robot manipulator is constructed using rigid links connected by joints with one fixed end and one free end to perform a given task. Relative motion between the adjoining links is enabled by the joints of the robotic manipulator which are the movable components of the structure. The other types of joints include linear joints that enable non-rotational motion between the links, and rotary type joints that ensure relative rotational motion between the adjacent links.

The important applications of robotic manipulator are:

- Planning Motion
- Remote Working
- Micro Bio Robots
- Humanoid robots
- Machining of tools.

In a mechanics context the degrees of freedom is defined as modes in which a mechanical device or system can move. The number of degrees of freedom is equal to the total number of independent displacements or aspects of motion. A robotic arm may have more than three degrees of freedom even if it is operating in two or three dimensional space. Thus the degree of freedom defines the motion capabilities of robots.

Let us consider a robot arm built to work like a human arm. In this case the shoulder motion can happen as pitch (up down), yaw (left and right) or rotational while elbow motion can only take place as pitch. On the other hand motion of the wrist can occur as pitch, yaw and also rotation (roll).

Such a robot arm has five to seven degrees of freedom. If a complex robot has two arms, the total number of degrees of freedom is doubled. Additional degrees of freedom exist in the end effectors, the legs and the head.

In our experiments we used the TAL Brabo Robot. Tal Brabo is 5 DOF robotic arm developed at TAL Pvt. Ltd.

This project aims at development of software (driver) and hardware interface (controller) for controlling robot using ROS.

1.2 EXISTING TRIO MOTION CONTROLLER

To create optimized processes and operations using the robot, we need to continuously communicate with the Robotic Manipulator. The information about the control actions and movement commands should be communicated to the controller of the manipulator with high speed and accuracy. TrioPC motion Controller is used to control the servo motors of the 5-DOF Brabo! manipulator. This controller does allow users to control the robot using inverse kinematics. Rather, users can only enter the co-ordinates in 3D workspace where the robot has to be moved, and the controller calculates joint angles and the trajectory taken to reach the desired co-ordinates. This is a major hindrance in creating automated systems like pick and drop etc. The reason being one cannot manually tell the controller what trajectory should be followed. For instance, let’s say the end effector of the robot has to be moved from Point A to Point B. There are infinite number of ways to move the robot from A to B if there is no obstacle in between as shown in Fig. 1.1. But in case of obstacles (Fig. 1.2) some new custom trajectory should be taken by the robot hence it required for the controller to understand this and implement intelligently. Secondly, in conventional robots like Kuka, ABB, drivers are written for a robotic manipulator’s controller, providing support for all operating systems so that they can recognize the physical components connected to the computer. However the TAL BRABO! Manipulator’s controller does not communicate with the computer through drivers, instead it utilizes an ActiveX component to communicate with it. ActiveX is a software framework that can be embedded into almost every application and it understand all programming languages. The TrioPC ActiveX component provides a direct connection to the Trio MC controllers via a USB or Ethernet link. The ActiveX component is developed by Microsoft and hence we were limited to use only windows platform for our project. Only MATLAB can be used to communicate and send co-ordinates to the trio controller through this ActiveX component.

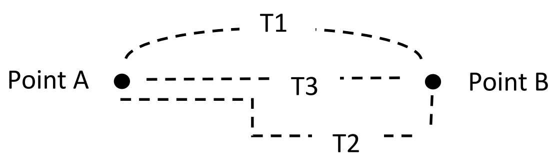


Fig. 1.1 Possible Trajectories to reach Point B

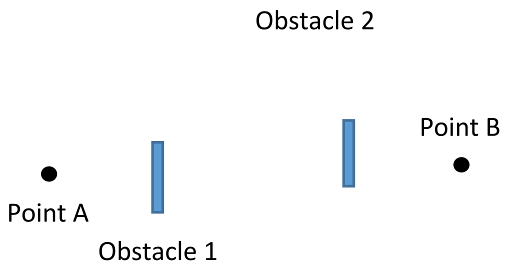


Fig. 1.2 Presence of Obstacles

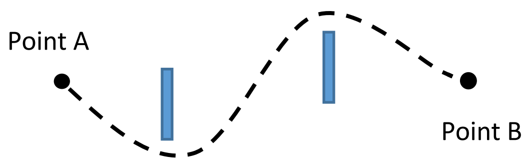


Fig. 1.3 Optimum Trajectory to avoid Obstacles

Hence, from the discussion above, it can be concluded that the two major drawbacks of the existing controller are:

1) No scope for modularity
2) Difficulty in communicating with the controller.

These drawbacks motivated us to come up with a new controller on which we can have full control. The controller should be robust enough to overcome the above mentioned difficulties.

This project is about controlling a 5-DOF manipulator arm (TAL BRABO) via ROS.
This involved following steps

- Controlling motors of robot using Arduino.
- Developing ROS node (driver) to communicate with Arduino based on joint position streaming.
- Designing pipeline for object follower using a USB camera attached to end effector of robot to demonstrate working of complete system.

1.3 ROS: ROBOT OPERATING SYSTEM

For the development of robot software, Robot Operating System (ROS) middleware was developed. Even though ROS is not an operating system, it provides services which are designed for a diverse computer clusters such as low-level device control, message-passing between processes, implementation of commonly used functionality, hardware abstraction and package management. The most important aspect of ROS is the fashion in the software runs and communicates with the hardware. It enables complex software and hardware interconnection without requiring to know how certain hardware works.

ROS provides a way to connect a network of processes (nodes) with a central hub. Nodes can be run on multiple devices, and they connect to that hub in various ways.

Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. [1]

Hardware Abstraction means encapsulation of hardware details via standard functions to provide safe access to control any hardware. Driver is a piece of software used to provide software level control over hardware. This process eliminates the need to write routines which are dependent on hardware as one can use drivers built by users/community or company delivering the product. This concept is very helpful for researchers so that they can focus on their research algorithms instead of diving into hardware level programs.

1.4 TIMERS AND INTERRUPTS IN ARDUINO

In Arduino or any micro controller a timer module is used to measure precise timing. It is basically a clock which is used to measure time events. The controller of the Arduino is the Atmel AVR ATmega168 or the ATmega328. They have 3 timers which are known as timer0, timer1 and timer2. There are two 8 bit timers (t0 and t2) and one16bit timer (t1). The 8bit and 16bit timers differ is their resolution. There are only 256 values available for a 8 bit timer while for a 16 bit timer there is a possibility of 65535. Thus 16 bit timer has a higher resolution. On the other hand the Arduino Mega series also has two variants of controllers which are the Atmel AVR ATmega1280 and the ATmega2560. Both of them only differs in memory size and are identical in other aspects. These controllers have 6 timers: timer 0, timer1 and timer2 are identical to the ATmega168/328. The timer3, timer4 and timer5 are all 16bit timers, similar to timer1 earlier. All the timers depend on the system clock of the Arduino system which is usually16MHz.

With the help of timer registers it is possible to configure these internal timers. In the Arduino firmware all timers are configured to a 1 kHz frequency and interrupts are generally enabled.
These timers are required to generate precise pulse signals for controlling motors i.e. for generation of PWM signals. An interrupt is an external event that interrupts the running program and runs a special interrupt service routine (ISR) [5]. Interrupts are used for implementing parallel routines in micro controllers. In this project, Serial Interrupt is used to read command data without disturbing the pulse generation process.

CHAPTER 2

SYSTEM ARCHITECTURE OF THE CONTROLLER

2.1 INTRODUCTION

A controller was designed to replace the existing TrioPC motion controller. We used five Arduinos to control 5 motors of the robot and an Arduino Mega to send commands to these Arduinos. Feedback was obtained from homing sensors to identify the joint limits. A layer wise design was chosen for modularity and efficiency.

2.2 TAL ROBOT HARDWARE SPECIFICATIONS

TAL robot is one of the procurement of TCS from TAL enterprises – the first and only manufacturer of industrial robot in India. It can perform various operations like Mig Welding, Pick and Place, Insert and Open, Copying the motion etc. The robotic hand with 5-axis of rotation and can carry a load of 10 kg. Base (1st Joint), Shoulder (2nd Joint) and Arm (3rd Joint) are controlled by AC powered Motors and Drivers while Wrist (4th Joint) and Hand (5th Joint) are controlled by DC powered Motors and Drivers. A controller/ software driver was designed by few students at TCS Innovation Lab. We were inspired by their design and have tried to implement similar hardware in our lab to bypass the controller. There are two layers to this. The green part is the hardware related to robot which needs to be controlled while the blue part represents the hardware and software designed by us to control the robot.

2.2.1 Homing Sensors

Robot has five homing sensors, one for each joint. These sensors have arc type shape. Sensor values are based on whether joint is in limit of arc or outside arc. These values are measured by Arduino Mega digital input pins. After some manipulation (to keep consistency among sensor readings) this data is sent back to ROS. Homing sensors works on 24V thus a simple voltage divider built using resistors to drop voltage level to 5V.

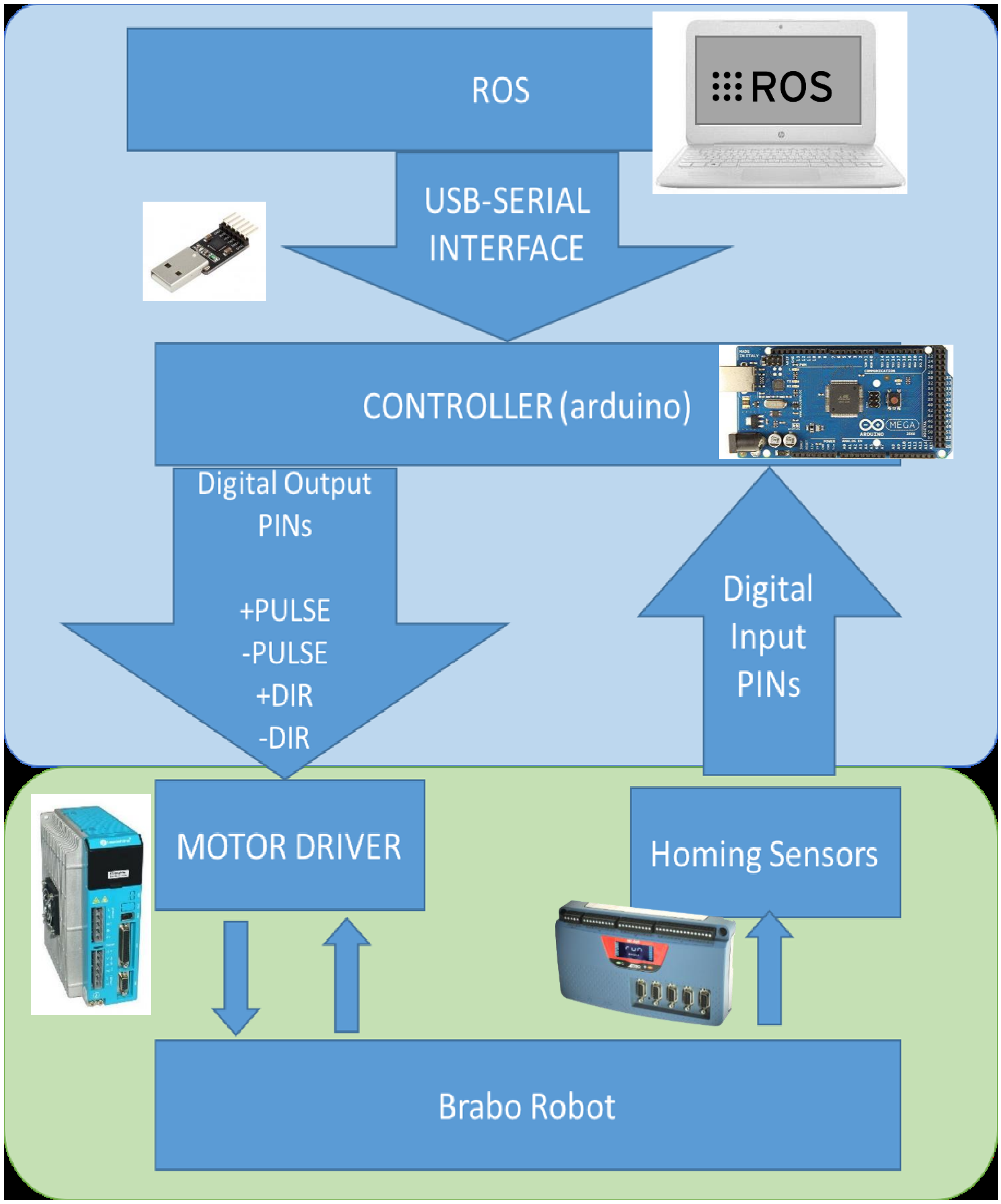
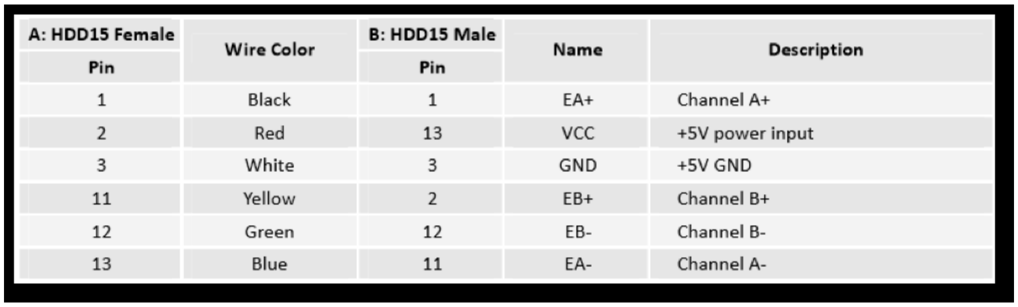


Fig. 2.1 Architecture of New Controller

2.2.2 Motor and Motor Driver Specifications

Motors installed in robot are step servo motors built by LEADSHINE company. 1st, 2nd and 3rd joints uses ES-MH series step servo motors with holding torque ranging from 8 Nm to 20 Nm. While 4th and 5th joint uses ES-M series step servo motors with Holding Torque ranging from 1 Nm to 8 Nm. Motor have two cables, one is encoder cable that is connected to the motor driver to provide feedback for working of internal PID control. While another is simply used to control motor. Motors also have a brake system which is activated when power is off to avoid falling of robot on ground. EB+ and EB- are brake connection. VCC and GND are fixed supply.
EA+ and EA- are control input.



A: HDD15 Female	Wire Color	B: HDD15 Male	Name	Description
Pin		Pin		
1	Black	1	EA+	Channel A+
2	Red	13	VCC	+5V power input
3	White	3	GND	+5V GND
11	Yellow	2	EB+	Channel B+
12	Green	12	EB-	Channel B-
13	Blue	11	EA-	Channel A-

Fig. 2.2 Motor Wiring Details

Leadshine's ES-DH2306 and ES-DH1208 drivers and ES-MH series step servo motors are used for 1st 2nd and 3rd motors. While Leadshine's ES-D508, ES-D808 and ESD1008 drives ES-M series step servo motors are used for 4th and 5th joints. The system includes an easy servo motor combined with a fully digital, high performance easy servo drive. The position, velocity and current loops are closed in real time by the internal encoder of the motor similar to other servo systems. The best servo and stepper motor functions are combined to increase the accuracy. Figure 2.3 shows control loop of motor driver.

2.2.4 Control Signal Required by Motor Driver

Motor driver require pulse and direction signal for moving motor. Motor move one step with one pulse based on direction pulse. To make a reliable operation, the ES drive requires the control signals to meet the setup time requirements as shown in figure 4. Otherwise losing of steps may happen. Motor driver can be connected by a male 44pin D-sub connector. To control motor it only requires 4 pins shown in later chapters.

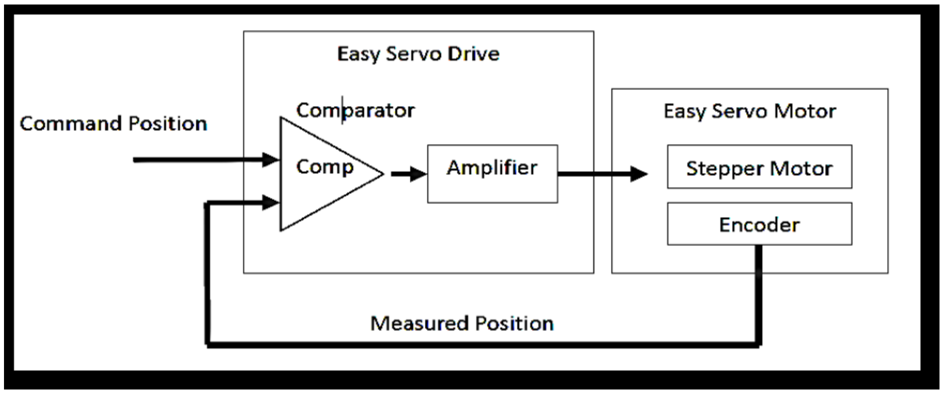
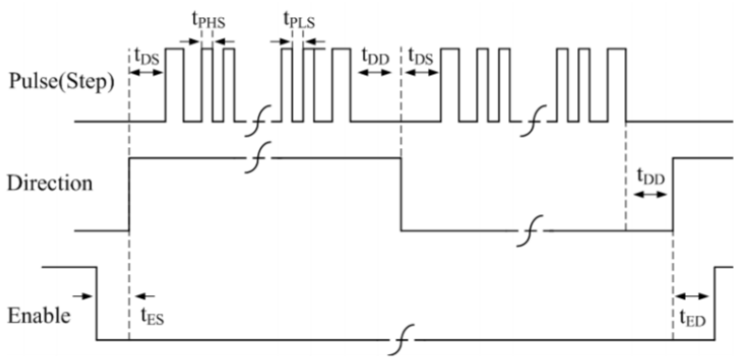


Fig. 2.3 Control Loop inside the motor driver



Symbol	Description
t_{DS}	Direction Setup Time
t_{PHS}	Pulse High Level Setup Time
t_{PLS}	Pulse Low Level Setup Time
t_{DD}	Direction Delay Time
t_{ES}	Enable Setup Time
t_{ED}	Enable Delay Time

Control Signal Setup Time						
Drive	Frequency	tDS	tPHS / tPLS	tDD	tES	tED
ES-DH2306	200K	>5uS	>2.5us	>5uS	>50ms	>50ms
ES-DH1208	200K	>5uS	>2.5us	>5uS	>50ms	>50ms

Fig 2.4 Control Signal Setup Timing

CHAPTER 3

LAYER 2: CONTROLLER ARCHITECTURE AND ALGORITHM

3.1 ARDUINO MEGA: MASTER CONTROLLER

The main aim was to design an embedded controller which reduces vibration of the motors. Each motor is controlled by different Arduino via motor driver. Motor driver accepts pulse input to increment position of motor as well as direction signal to set direction of rotation. Overall architecture of controller is as shown in figure 6. Arduino Mega is used as central controller which receives command from ROS and then sends it to other Arduinos. It is also responsible for collecting homing sensor data and sending it back to ROS. Each Arduino receives number of pulses (based on angle to move) and direction. These commands are then processed and precise and continuous pulses are generated using internal Arduino TIMERS which are sent to motor driver.

3.2 HOMING SENSORS: FEEDBACK

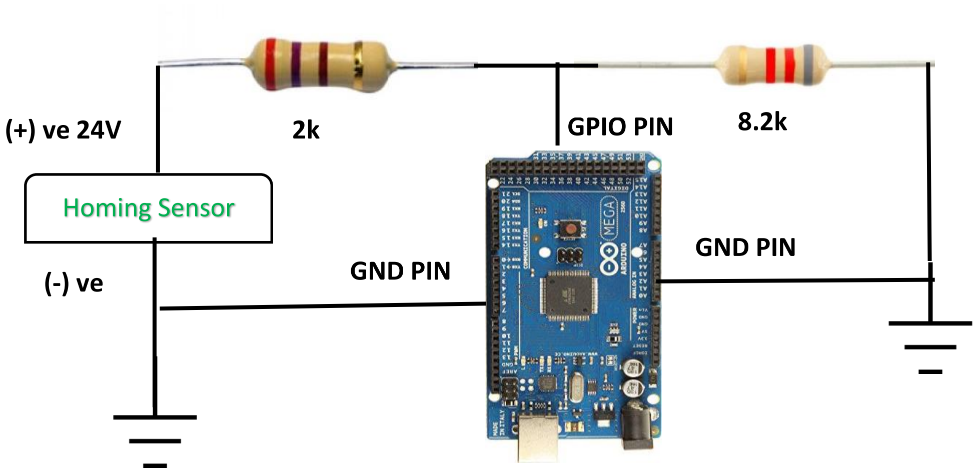


Fig. 3.1 Voltage Divider for Homing Sensor

Vout = Voltage at GPIO Pin = 24V * [2k / (2k+8.2k)] = 4.8 V

There are five homing sensors located on each motor. When the motor rotates a certain angle the homing sensors react to it by either making their output high or low. These homing sensors are connected to the TrioPC Motion controller. Their output is 24V. But the Arduino GPIO Pins can only handle 5V. Hence we built a voltage to step down 24V to 5V.

Connect Homing Sensor of **X Joint to pin 33, of Y Joint to pin 34, Z Joint to pin 35**

of **U Joint to pin 36, of V Joint to pin 37 on Arduino Mega**

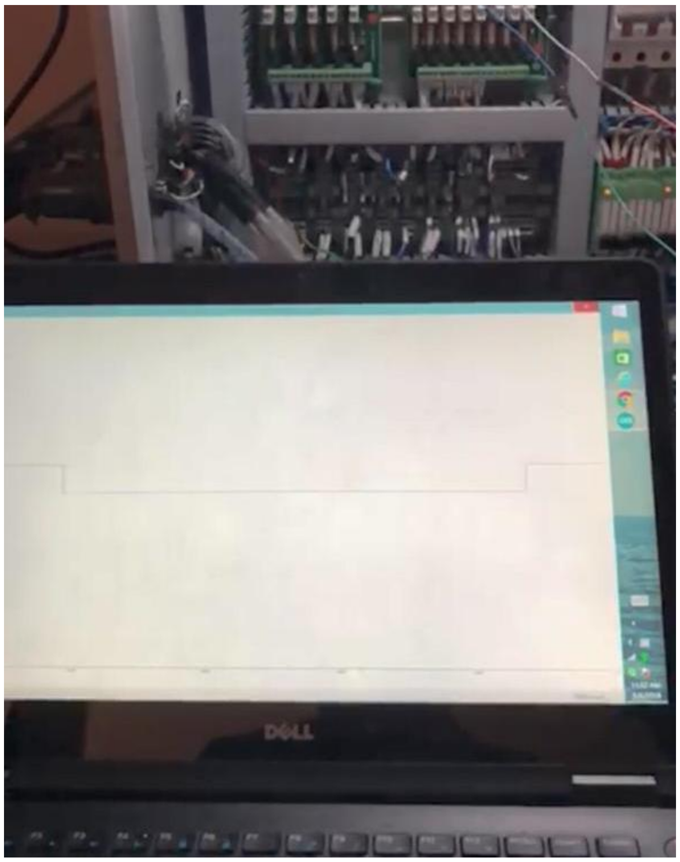


Fig. 3.2 Pulse output of the homing sensor

The output of the homing sensor is constant pulse unless it reaches the homing point. It can be seen in the figure that the pulse goes low once it reaches the homing point.

3.3 COMMAND SET SPECIFICATION

Command set sent from ROS consists of total 11bytes of data as described.

(B1, B2)	Number of pulses for motor 1
(B3, B4)	Number of pulses for motor 2
(B5, B6)	Number of pulses for motor 3
(B7, B8)	Number of pulses for motor 4
(B9, B10)	Number of pulses for motor 5

(B11) This byte is used as direction byte. 5 bits in this byte are assigned for direction of each motor.

3.4 PULSE GENERATION ALGORITHM IN ARDUINO

This section covers the most difficult and most important part of controller. As already explained before we need to generate pulses of precise timing in order to control motors. Streaming rate is 100 i.e. a command set is received by Arduino every 10ms. Pulse is generated by flipping digital pin whenever timer1 overflows. We set timer1 to measure half time period of pulse. There is a counter for number of pulses which is decreased after one pulse is generated and is updated when next command is arrived. Main point in this is that, the pulse generation is not stopped even if counter is reduced to 0 i.e. even if somehow all pulses are generated even before next command arrives Arduino keeps generating pulses. This is done to keep continuous pulses to reduce vibration. Thus in case more or less pulses are generated than required they are accounted in next command set. We update this counter based on next command set as well as previous number of pulses that were remained or extra. Reading command set is done using serial interrupt to make it a parallel process. When command set is read properly dataReaded flag is set. Arduino keeps on generating pulses until ROS sending commands or sends zero as number of pulses. In case ROS stops sending command it is detected after 50ms, i.e. Arduino waits for 50ms for command to arrive. In case no command is received a STOP_MOTION flag is set. This starts a stop motion routine which slowly reduces the speed of motor from current speed to complete halt and Arduino goes in waiting mode. As soon as new command is detected whole process start again.

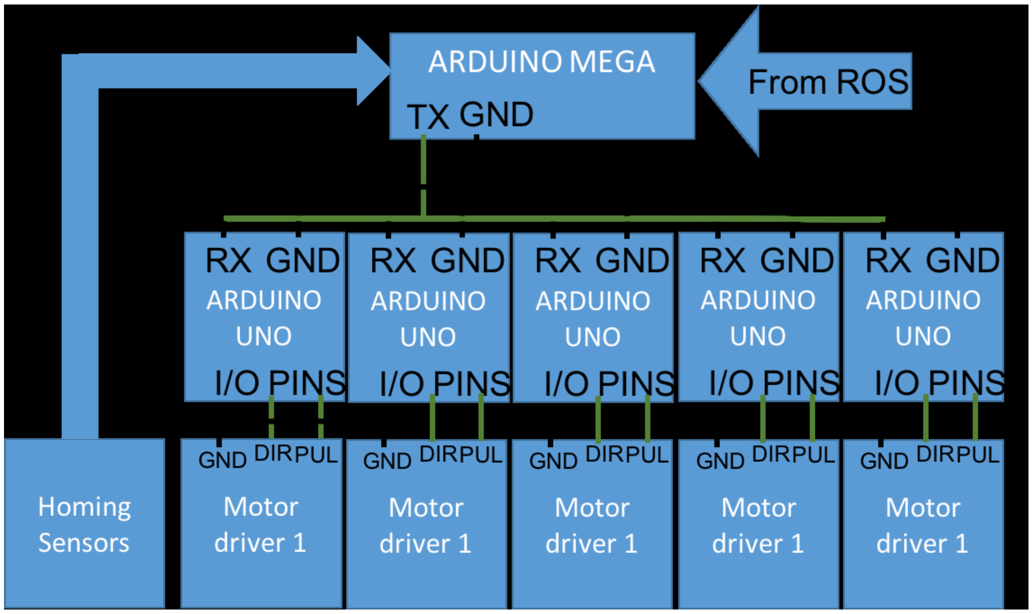


Fig. 3.3 Overall architecture of the controller

Thus, the user is sending a command from the laptop through the USB-Serial Interface to the ARDUINO-MEGA. The MEGA picks up signals from the Homing Sensors and sends it back to the user computer again through the USB-Serial Interface. Further, the MEGA also sends the 11 Bytes of Command Instruction received from user computer to the other five Arduino-UNOs through the Tx Pin of the Arduino Mega and the Rx Pin of Arduino Uno receives it. The Arduino –Uno is interfaced with the Motor Drivers as per the connections below. Only four connection are made from the Uno to the motor-driver. Additional four connections for the enable and alarm pins are tapped from the 44-pin D-sub Male Connector. Without the enable connections the controller will not e=be enabled.

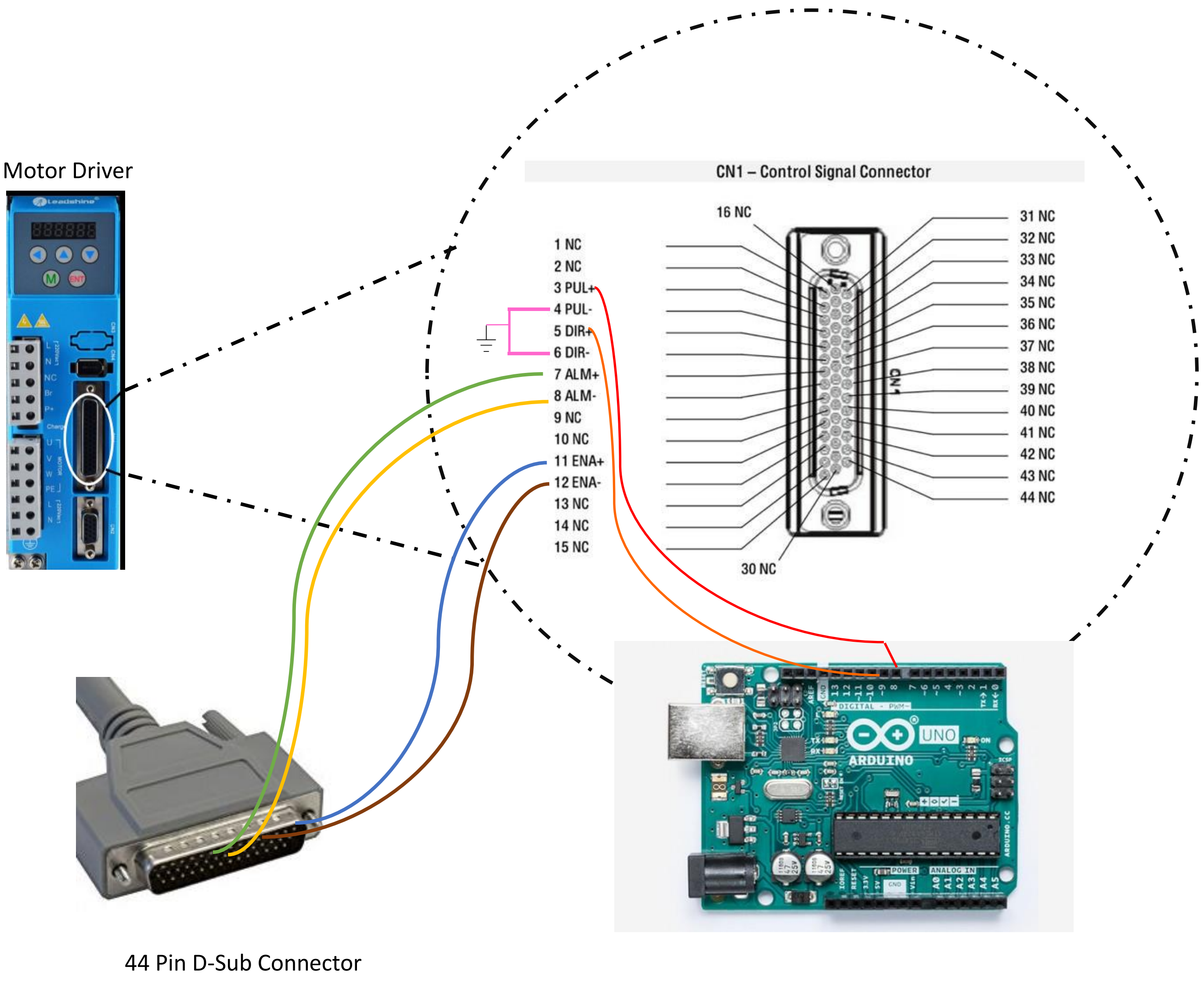


Fig. 3.4 Motor Driver Connection to Arduino UNO

PIN 8 on Arduino UNO = PUL+

PIN 9 on Arduino UNO = DIR+

CHAPTER 4

SOFTWARE INTERFACE AND CONTROL INSTRUCTION

4.1 INTRODUCTION

As it can be observed from the previous chapter, every motor requires two signals from the user. The two signals are the direction in which the joint should be moved and the pulse/step that should be given for the movement. The larger the step value, the more is the speed of the motor to achieve the desired position.

The pulse given to every joint can be limited to a maximum value of 100 but to safe its better to operate the robot with a pulse of approx. 20-30. The direction bit of motor has been computer manually. It is a 8 bit long information as follows:

Direction byte = 0b001vuzyx

The direction bit for moving X Joint in clockwise direction is 0b0010001

The direction bit for moving X Joint in counterclockwise direction is 0b0010000

The direction bit for moving Y Joint in forward direction is 0b0010010

4.2 ROS-SERVER

ROS is used to control Robot via USB-Serial Interface. The main node is ‘tal_brabo_driver’. This node is driver node and is responsible for interfacing ROS and Arduino. When this program is run, first a serial connection is established with the Arduino Mega. Then the user is asked to enter the joint positions of x, y, z joints which are to be manually determined. Every joint of the robot then reaches its homing angle and stops. After that all joints start moving together to come to their home position. The number of pulses required to home are manually computed and noted in the code. After attaining the homing position or initial position of the manipulator, the other nodes are created and the controller waits to receive further information on these topics.

4.3 POSITION STREAMING INTERFACE

We are using Position Streaming Interface for controlling robot. In this method, joint positions are streamed to the controller. Joint velocity is computed by the controller. ROS send command to controller at some fixed publishing rate (default is 100). This means we send command to controller every 10ms. Its job of controller to generate corresponding pulses and direction signal in a time frame of 10ms.

CHAPTER 5

VISUAL SERVOING AND KEYBOARD TELEOPERATION

5.1 KEYBOARD TELEOPERATION

The functioning and controlling of an articulated robot mainly consists of feeding the right signal to the right actuator at the right time. Collaboration between robots and humans means different things to different people in different applications. Collaboration could range from a robot and a human simply operating in the same area, to operations that require complex, interdependent decisions based on joint goals. Despite the level of coordination, all effective collaborations require understanding the control allocation processes, and human engagement or reengagement strategies. Thus manual control is required for the robot in many situations.

However, we have unintentionally disabled the Teach-Pendant control from the controller while implementing this new controller. Hence to give the robot the capability to work manually we introduced a keyboard tele-operation where one can control all the actuators of the robot using a simple laptop keyboard.

For this purpose we wrote a simple if-else loop. Using the python pygame library we record the key-stokes of the user on the keyboard and then based on the respective joints move. The available key presses to move a robot are:

X and Right Arrow Key (→)

X and Left Arrow Key (←)

Y and Up Arrow Key (⬆) etc.

5.2 VISUAL SERVOING TO DEMONSTRATE THE ENTIRE WORKING

The use of visual feedback to control a robot is known as visual servoing (Hutchinson et al.1996). Vision data is acquired from a camera mounted directly on a robot manipulator or on a mobile robot. The motion of the robot results in camera motion. The camera can also be fixed in the workspace such that it can observe the robot motion from a stationary position [8]. The accuracy of the so formed system depends on the accuracy of the robotic manipulator and the visual sensor. The accuracy of these subsystems can be increased by using 'visual feedback' control loop [7]. We implemented a Visual Servoing system and following are the steps for it.

We implemented a Visual Servoing system and following are the steps for it.

5.2.1 Tal_Driver_Node

After homing, the robot is waiting for further commands to move to a new desired position. The ROS graph structure for the same can be seen in figure below. Images from USB camera are grabbed by usb_cam_node which are further processed by object_tracker_node to approximate 3D coordinates of center of ball used as object to be tracked by robot end-effector. Then based on these coordinates and model of robot velocity_controller_node, computes the joint velocities and then updates joint angles of robot and publishes new joint angles at /joint_command topic. Based on joint angles published by velocity controller tal_brabo_driver decides what commands are required to be sent to Arduino and sends them at specific streaming rate via USB Seial Interface. The velocity controller can be replaced by any other controller based on task. Controller needs to publish joint angles on topic /joint_command topic at any rate smaller than or equal to STREAMING RATE of the tal_brabo_driver which is 100 in default settings.

This node is the main node responsible for sending commands to Arduino based on joint angles received. This node is subscribed to /joint_command topic having message type sensor_msgs/JointState. This node keeps track of joint angles, as there is no feedback available from robot about current state of robot. It is important to note that restarting this node will first reset robot to its initial position.

5.2.2 Node_Object_Tracker

This node detects green colored circular object in the image and calculates center of object. This node uses already built in method of blob detection in OpenCV Library. First image is thresholded based on HSV range of object. Then based on some threshold parameters best suited blob is selected and center is calculated. Node approximates Z distance as same as size of circular object. This approximation is valid only in case of circular objects. Since, velocity controller is based on PID therefore, there is no need to calculate distances in proper units which would require camera matrix and other things. Values of PID constants are tuned according to pixel level coordinates and size of object treated as distance in Z axis. We also implemented the Faster CNN Code written by a CS student to detect a tennis ball.

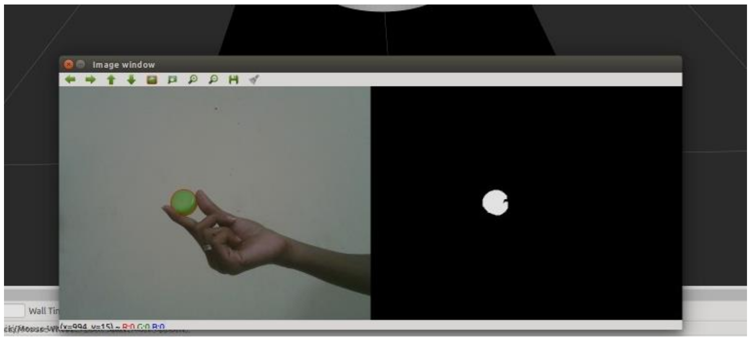


Fig. 5.1 Object Detection

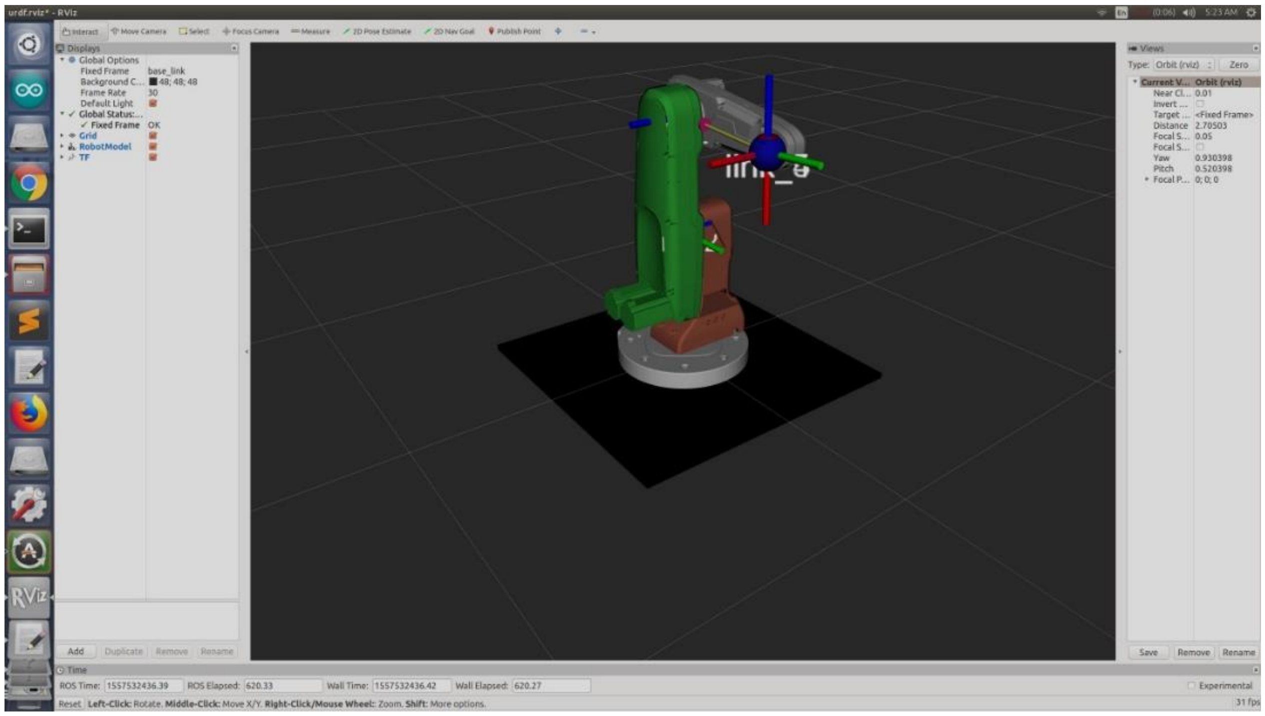
5.2.3 Velocity_Controller_Node

This node is responsible for publishing appropriate joint angles to tal_brabo_driver based on position of center of object tracked. This node is subscribed to ball_center to get update on center of object from object_tracker_node. This node is also subscribed to joint_states topic for getting feedback of current joint angles of robot. This node first calculates end effector velocity of robot using simple PID algorithm. The error in PID is defined as difference between desired location (0, 0, D) and current location of object. Desired location is set to (0, 0, D) as it will ensure robot will try to move in such a way that object remains at center of camera and distance D away from it. Thus in the case the user tries to move the object, the robot would try to follow the object. These end effector velocities are then translated to joint velocities using inverse Jacobian. Inverse

Jacobian is calculated based on current joint angles as received by this node. Target joint angles are calculated by adding these joint velocities to current joint angles. Finally, these target joint angles are published on topic /joint_command.

5.2.4 URDF Model

A URDF Model based on Xacro is made for the manipulator to initially test the code and then implement on the robot. Xacro is an XML macro language. With xacro you can construct shorter and more readable XML files by using macros that expand to larger XML expressions. The Universal Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot.



Global Options	
Fixed Frame	base_link
Background C...	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status:...	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
TF	✓

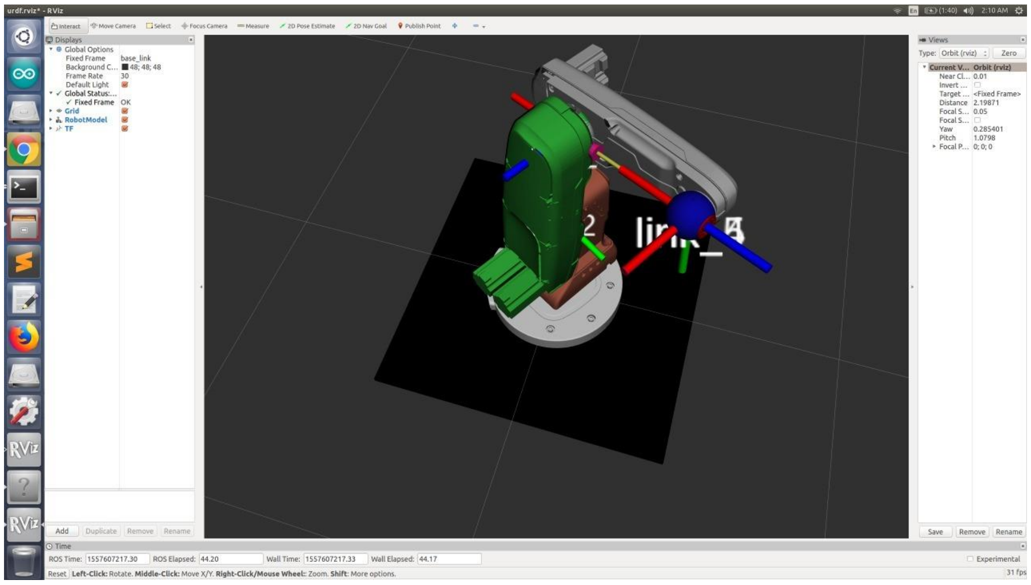
Type: Orbit (rviz)

Current V...	Orbit (rviz)
Near Cl...	0.01
Invert ...	☐
Target ...	<Fixed Frame>
Distance	2.70503
Focal S...	0.05
Focal S...	☐
Yaw	0.930398
Pitch	0.520398
Focal P...	0; 0; 0

ROS Time: 1557532436.39 ROS Elapsed: 620.33 Wall Time: 1557532436.42 Wall Elapsed: 620.27

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

Fig. 5.2 URDF Model in Rviz (a)



```
Global Options
  Fixed Frame        base_link
  Background C...    48; 48; 48
  Frame Rate         30
  Default Light      ✓
Global Status:...
  Fixed Frame        OK
Grid                 ✓
RobotModel           ✓
TF                   ✓
```

```
Type: Orbit (rviz)
Current V...   Orbit (rviz)
  Near Cl...   0.01
  Invert ...   ☐
  Target ...   <Fixed Frame>
  Distance     2.19871
  Focal S...   0.05
  Focal S...   ☐
  Yaw          0.285401
  Pitch        1.0798
  Focal P...   0; 0; 0
```

```
ROS Time: 1557607217.30  ROS Elapsed: 44.20  Wall Time: 1557607217.33  Wall Elapsed: 44.17  ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.  31 fps
```

Fig. 5.2 URDF Model in Rviz (b)

5.2.5 Algorithm Steps

1. First serial connection is made with Arduino Mega via USB.
    - User inputs are taken to get initial position of joints. Since homing sensor could only detect whether joint angle is on magnetic arc or not, these does not give exact position. Therefore user is asked to input joint location of first 3 joints, whether they are on black or white side of arc (Sides are marked on physical robot).
    - Homing procedure starts
2. Based upon user input joints are slowly rotated to achieve initial position (when joint are on one edge of arc). Joint 4 and joint 5 are rotated in full circle to reach edge of arc of homing sensors.
3. After homing robot the command is to reach a different initial position required for task of object following.
4. After this, the node starts waiting for messages on the topic /joint_command to get target joint angles.
5. As soon as it starts receiving commands it calculates pulses to be sent to Arduino based on GEAR RATIO and angle to be moved. Also there is limit to maximum number of pulses to be sent to Arduino for safety reasons. Directions of motors are calculated and finally a

command set is transmitted. Also current joint angles are updated and published. Since motor4 and motor5 are connected to each other by a bevel gear, thus actual joint velocity of joint5 is dependent on joint4 velocity. Thus pulses are provided in such way to compensate for this. This compensation is based on following observation, Joint5 provided commanded is equal to joint5 desired velocity minus joint4 desired velocity.

5.2.6 Steps To Operate

1. Switch on power supply of robot.
2. Start controller button on control box on robot.
3. Connect Arduino's and power them
4. Start tal_brabo_driver node (tal_brabo/tal_driver.py)
5. Enter positon of first 3 joints (white or black i.e. 0 or 1) **(Always Enter Zero)**
6. Wait for homing to be completed
7. Now you can start other nodes
8. Attach camera to end effector
9. Start usb_cam_node
10. Start object_tracker node (tal_brabo/object_tracker.py)
11. Start Rviz (brabo_description/launch/display.launch)
12. Start velocity controller node (tal_brabo/velocity_controller.py)

- ❖ SAFETY MEASURES

1. Always start controller button on the box for reducing vibration in motors (Green Button)
2. Streaming rate to Arduino is 100. Thus any message published on the topic '/joint_command' would result in publishing a command to Arduino. Arduino would try to achieve desired angle in 10ms. Tal_brabo_node keeps track of current joint angles of the robot. When any node publishes to '/joint_command' topic tal_brabo_node calculates velocity based on published angles and current angles and accordingly sends command to Arduino. This velocity will be high if difference between current angles (tracked by

tal_brabo_driver_node) and angles published on '/joint_command' is large as time step is only 10ms. Tal_brabo_driver node publishes current joint angles on the topic '/joint_states' so that other nodes can use that to publish desired angles. Therefore always use current angles published on topic '/joint states' to publish desired joint angles. Otherwise a high velocity would result in safety problems.

3. Do not start publishing in between task, always restart tal_brabo_driver node before starting to publish desired joint angles. Because failing to do so would result in same problem described in above point.
4. When you write a new code, do all the above mentioned steps but don’t connect actual motors directly. Just connect Arduino Mega and always first run the code on URDF model in R-Viz and check if robot moves as expected by your code and only then implement on the actual physical robot.

5.3 RESULTS AND DISCUSSION

Robot was able to follow ball and whole system worked for prolonged time without any single system failure. Visualization on Rviz was in good sync with the real robot in all tasks. Following tasks were successfully completed by robot:

- Homing at given position
- X-axis motion
- Y-axis motion
- Z-axis motion
- Pure rotation of end effector
- Ball follow in 2-d plane
- Ball follow in 3-d

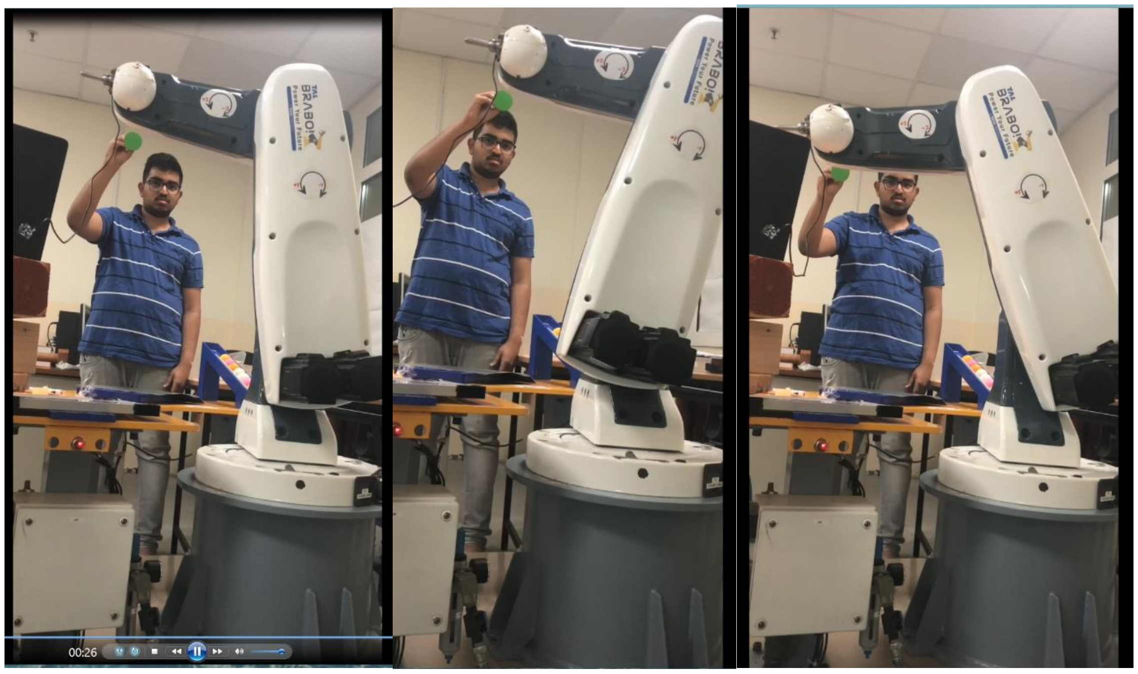


Fig. 5.3 Results of Real Time Visual Servoing

5.4 FUTURE WORK

There are many advantages of using robotic machining cells over other conventional machines like the Computer Numerical Control machine especially while considering the much wider working area which allows it to accommodate much bigger workpieces and the flexible kinematics of robotic arms, are often capable of machining parts with intricate details and complex shapes. Industrial robots are thus automated and programmable. Despite these obvious wins for robotic machining, there is some common criticism about the controlling of the robots. It is usually done by using a standalone computer that communicates with the controller of the robot. However, controlling the dynamic motion using this method requires expertise in coding and the domain in general. Since Internet of Things (IoT) is a rapidly growing technology, it can be integrated with the robot to make its operation and control user-friendly.

REFERENCES

[1] Robot Operating System: https://en.wikipedia.org/wiki/Robot_Operating_System

[2] ROS Introduction: http://wiki.ros.org/ROS/Introduction

[3] ROS Concepts: http://wiki.ros.org/ROS/Concepts

[4] ROS Higher Level Concepts: http://wiki.ros.org/ROS/Higher-Level%20Concepts

[5] Arduino 101 Timers and Interrupts: http://robotshop.com/letsmakerobots/arduino-101-timers-and-interrupts

[6] Hans, Sikander. (2018). A Brief Comparative Study of Visual Servoing Systems. 6. 2321-0613.

[7] Carlos lopez-Franco,Nancy Arana-Daniel and Alma Y.Alawis ., " Visual Servoing on the Sphere Using Conformal Geometric Algebra",2012,Adv. Appl. Clifford Algebras Springer basel, AG DOI 10.1007

[8] Brian C. Becker, "Active Guidance for Laser Retinal Surgery with a Handheld Instrument",2009, 31st Annual International Conference of the IEEE EMBS Minneapolis, Minnesota, USA,pp.5587-5590.