A ROBOTICS FRAMEWORK FOR SIMULATION AND CONTROL OF A ROBOTIC ARM FOR USE IN HIGHER EDUCATION

by

Craig Christensen

A THESIS DEFENSE

Presented to the Faculty of

The School of Computing at the Southern Adventist University

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science–Embedded Systems

Under the Supervision of Tyson Hall

Collegedale, Tennessee

May, 2017

A ROBOTICS FRAMEWORK FOR SIMULATION AND CONTROL OF A ROBOTIC ARM FOR USE IN HIGHER EDUCATION

Craig Christensen, M.S.
Southern Adventist University, 2017

Adviser: Tyson Hall, Ph.D.

Robotic arms have been in common use for a several decades now in many areas from manufacturing and industrial uses to hobby projects and amusement park rides. However, there have been very few attempts to make an inexpensive robot arm with a software stack for use in higher education. This paper will outline a control and interfacing software stack built on the Robot Operating System (ROS) and a simulation of the 5 degree of freedom (DoF) robotic arm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots can be used to teach a wide number of classes and concepts including basic programming, concurrent programming, dynamics and control, mechatronics, engineering, electronics, forward and inverse kinematics, computer vision, and path planning [1, 2, 3, 4]. Currently there are few to no standardized robotic arm platforms with ready-made software stacks able to use both physical and simulated robots for teaching in higher education.

The platform introduced in this paper will allow students to implement and test additions and modifications to the software on their own computers through the simulation environment, then test their code on the physical robot, without any need for translation or modification of their code to make it work in the real world. Using practical examples has been shown to be beneficial to learning, and especially helps to interest students in the subject and aid in the self-learning process.

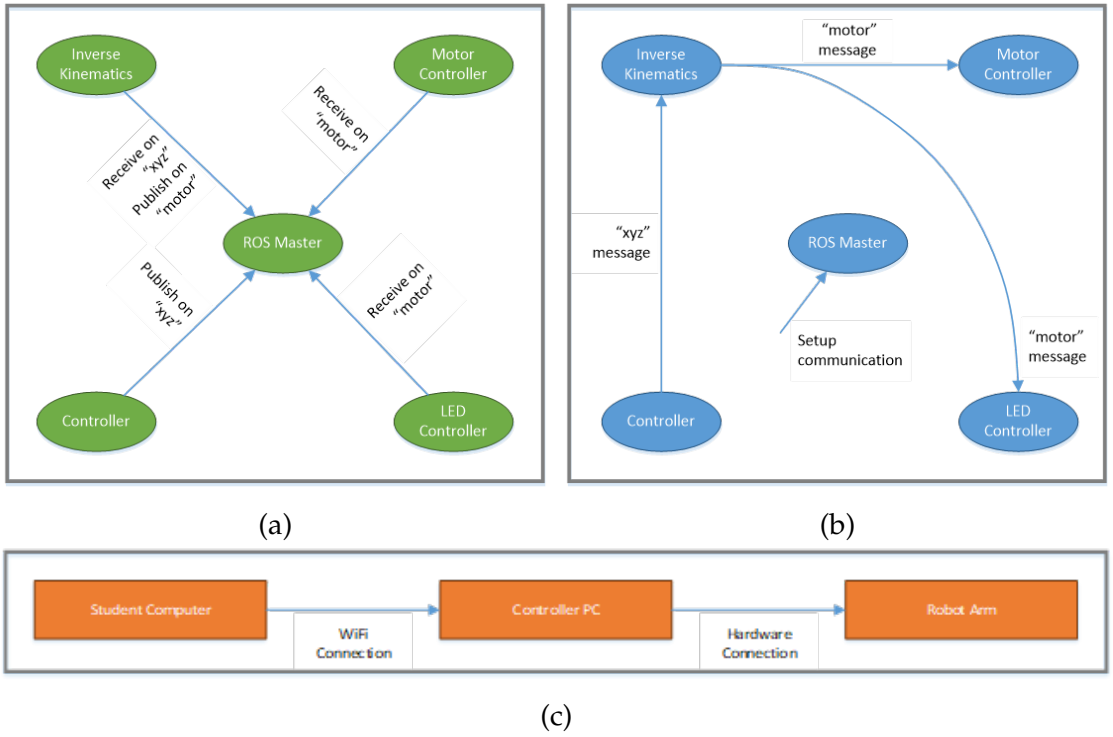

Figure 1.1: (a) When the ROS system first starts up each node connects to the ROS Master and gives it messages the node publishes and receives on [5]. (b) Each receiving node will then connect directly to the sending nodes for message passing [5]. (c) The system will be implemented on a computer running Ubuntu 16.04 and take input from some source to control the robot arm [5].

1.1 Goals and Requirements

The software control stack for the project is built on the Robot Operating System (ROS). ROS has become a popular robotics platform for use in many types of robots [5]. The ROS platform is detailed in Figure 1.1. The system consists of two parts: the first part is the nodes of the system shown in Figures 1.1a and 1.1b. Nodes are the programs of the system and implement hardware integration, services, and interaction with the outside. Nodes may be custom built for the application or may come from the ROS community allowing hardware integration and other non-focus items to be done faster. The second part of the system is

message passing, and each node is able to publish and receive messages. ROS controls the message passing between nodes and sets up the interaction between nodes as shown in Figure 1.1a. After the initial setup of the communication, messages are passed directly from one node to the next, as shown by the blue graph in Figure 1.1b.

The system of nodes and messages makes any software stack created on ROS a distributed system allowing for implementation flexibility. The functionality of the project is implemented in the control of both a 3D simulation of and physical 5 DoF SCORBOT-ER III robotic arm. Students can implement or reimplemented a function of the system by adding or recreating a node, leaving the rest of the system fully functional.

The functionality and movement of the robot is shown through pre-programmed tasks such as performing the Towers of Hanoi, it is also able to perform real-time movement based on inputs from an API.

1.2 Motivation

This project will allow science, technology, math, and engineering (STEM) students to work on a simulated and physical robot in class. The final product will make robotics learning and research easier by providing a robot and a software stack ready-made for immediate use and modification as opposed to each university and class starting from scratch. This allows the class time and projects to focus on learning objectives rather than the nuts and bolts of the specific system.

Chapter 2

Background

2.1 Education

Robots can be used to great effect in the area of education, and recently a greater emphasis has been placed on the subject due to a focus on science, technology, engineering, and math (STEM) especially in lower education [1]. Proposals of programs to introduce robotics into classrooms are very common. Unfortunately most use solely a simulated environment, or a physical robot, but few use both. A purely simulated approach can be dissatisfying to students, as seeing a finished final physical product provides a great deal of motivation and tangibility to the experience making classes engaging and attractive [6, 1]. A purely physical system can also be at a disadvantage, as students may only be able to work on the robot while they are physically in the lab. A physical teaching tool can have a large effect on the learning process, and if used along side virtual systems they both can provide a lab environment that engages students in the learning process [7, 8, 3, 9, 10].

Robotic arms specifically have been used to teach in the areas of basic program-

ming, concurrent programming, dynamics and control, mechatronics, engineering, electronics, forward and inverse kinematics, computer vision, and path planning [1, 2, 3, 4].

Many examples of robots designed for use in a classroom environment have been proposed for use in lower education teaching. Some of the most basic robots have been used in the area of early education[9, 11, 12]. These robots many times use graphical programming languages as is the case of the AERobot designed by Rubenstein et. all [9]. This robot was designed to be complex enough to do interesting tasks, such as obstacle detection and line following, but cheap enough to be affordable in mass to any community [9]. Another example of robots designed for lower education are the LEGO NXT robots used in the FIRST LEGO League competitions [11, 12]. These robots are used in competitions for kids from elementary to high-school, and allows them to design and program the robots to accomplish tasks in the competition [11, 12]. Though these robots are useful for teaching entry level programming and for introducing the concepts of robotic programming, they do not focus as much on the algorithms of the robot or how the robot actually functions, only the higher level functionality.

The second area for use of robots is in universities, colleges, and other forms of higher education. This demographic usually focuses more on the intricacies of robot construction, control algorithms, such as forward and inverse kinematics, and sensor processing. CHARM is a robotics course developed just for this purpose by Singh et. all [2]. This course teaches students how to design and program a robot for coin sorting, and in the process teaches kinematics, path planning, and robot vision through a coin sorting project [2]. Though this robot is good for teaching a number of concepts it only uses a three degree of freedom (DoF) SCARA robot arm. This makes the robot good for teaching in two dimensions,

but does not allow for three dimensions. The robot may also be hindered in a lab environment because it does not propose a framework to build on requiring students to write the entire control system every time rather than focusing on the algorithms of the system one at a time, therefore a bottom up approach must be used, and the robot is only useful for the one class, not for a range of classes.

2.2 3D Printing

3D printing has quickly become a common method of rapid prototyping in recent years [13]. This is due to advances in 3D printing technologies and standardizations making more wide spread use possible [13]. 3D printers are now common in universities for use in research and product design. This makes the method ideal for use in producing products in a distributed fashion for education. 3D Printing has already been used for robotics research in universities as is the case in the 3D printed hand by Mukhtar et. all [14]. Poppy is another 3D printed platform designed for education by Lampeyre et. all [15]. Poppy has been used in several research projects at various universities since its design [16, 17].

Several robotic arm designs are available for free from various sources and makers, many of which are open-source and modifiable to meet the needs of the user. There are many simple robot designs available, but these designs have severely limited degrees of freedom [18, 19, 20]. There are several robot designs with more degrees that would be better suited for a general purpose robots. One such robot is the five axis Thor robot by AngelLM, unfortunately a five axis robot may not be sufficient enough for more advanced classes [21]. Another advanced design is the Zortrax robot, this robot looks good and is a simple sleek design, unfortunately not all of the joints are motorized making it unusable for control

classes [22]. The Moveo arm by BNC3D is a good viable option as a six axis fully designed robot arm, this would make a good robot for use in a class environment [23, 24]. The last 3d printed robot design is by Andreas Hölldorfer [25, 26].

2.3 Simulation and Control Software

Simulation software is very common for a variety of different robots. Several popular robotics simulators exist [27]. These systems include the Virtual Robotics Toolkit, Robot Virtual Worlds, RoboDK, Microsoft Robotics Developer Studio, Webots, and Gazebo [27].

Both the Virtual Robotics Tool Kit and Robot Virtual Worlds simulators are designed for use with LEGO Mindstorms, Vex Robots, and other LEGO based systems, and primarily target early STEM education [27]. The systems are not as flexible as others due to their limited target audience, and though they have been used in higher education, the platform is aimed at first time programmers [28, 11, 29, 30, 31, 32]. The limits of the system force any modification to the programming and control of the it to be done in round about methods with external controllers [31]. The RoboDK, this software is designed for use in industrial robots and does not allow for externally controlled or open platform robots [27]. Microsoft Robotics Developer Studio has support for a wide array of platforms and is supported by a large company [27]. Though this has been a well received platform, support for the software has been canceled, meaning that any knowledge gained using the platform will be useless as the students go on to future projects [27]. Webots and Gazebo both officially support the ROS platform, and can be used to simulate any robot design [27]. Webots is a closed source paid platform, and therefore the system cannot be as modified for specialized uses [27, 33]. Gazebo,

in contrast, is a free, open-source platform and has been used in developing interactive robots extensively [27]. Gazebo is also can be compiled and run on Linux, OSX, or Windows [34]. The final simulator available is RViz [35]. RViz is packaged with ROS in the full desktop installation [35]. RViz works only on Ubuntu Linux, and does not have support for other operating systems [35].

There are several different software stacks available for robot control. Many of the software stacks are custom and designed for a single robot. This is the case for systems such as BRACON by Rivas et. all [36]. If the software is not widespread enough as an open source software, or is proprietary, it may not allow for the best use in education as it does not allow for modification for new systems. Many of these systems only allow limited methods of interaction, such as through gCode, that is simply a communication format, and does not allow dynamic movement or feedback.

The Robot Operating System (ROS) is a promising new open source robotic framework for use in a wide array of robots. ROS has been used in previous educational robotics projects such as Nelson [37]. Other projects have focused on extending ROS for use in MATLAB such as in the case of ros4mat [38]. ROS comes with both control software and a simulation environment that can be controlled simultaneously. The software is modular, as shown if Figure 1.1 and can be distributed across devices [5].

Chapter 3

Implementation: Southern Arm Control

The Southern Arm Control (SAC) project consists of includes seven ROS packages, as well as, a setup project to install ROS Kinetic, its components, and the SAC system.

3.1 Robot Operating System (ROS)

ROS allows for an open-source, easily distributable system that is freely available to students [5]. The base of a ROS project is a workspace that acts as a container for the packages of the project [39]. ROS packages are sub-containers within the project that hold the code for nodes, messages, descriptions, and all other parts of a ROS system [39, 5]. For message passing between nodes, ROS has two types that the SAC project uses:

- Messages – Used to pass information one way from a publisher to a subscriber [40].

Package Name	Description
gazebo8_ros_pkgs	Used for integrating Gazebo and ROS.
gazebo8_ros_control	Used for integrating Gazebo and ROS.
ros-control	Used for controlling the robot model.
ros_controllers	Used for controlling the robot model.
moveit	Used for picking up and interacting with items in the world.
gazebo_grasp_plugin	Allows objects to be attached to the robot for pickup and movement.

Table 3.1: The packages used in the SAC system. The MoveIt! package is not currently in use in the SAC system.
[43, 44, 45, 46, 47, 48]

- Service – Used to pass information to and receive a response from a service [41].

Third-party packages can also be distributed and installed into ROS, in which case, they will reside in the ROS installation directory. The SAC system is built on the ROS platform. The system uses six external ROS packages, listed in Figure 3.1, ten custom packages, listed in Figure 3.2, and a setup project for setting up ROS and the SAC system. This is detailed in Section 3.8 [42].

3.2 Messages

ROS messages are used to pass information between nodes. ROS includes several basic message types divided into packages:

- Standard Messages – includes messages for basic types such as integers, floats, and booleans [58].
- Common Messages – includes packages for geometric, sensor, diagnostic, navigation, and action library messages [59].

Package Name	Description
sac_controllers	Holds all of the controllers for the SAC system as described in Section 3.3.1.
sac_description	Contains the robot model for use in Gazebo.
sac_drivers	Holds the drivers for the system to interface with external robots. See Section 3.3.3.
sac_gazebo	Holds the world to display the robot and other models in Gazebo.
sac_launch	Holds the Global launch files to start the system.
sac_translators	Holds the nodes for translating inputs to joint angles space goals as detailed in Section 3.3.2.1.
sac_msgs	Contains the definitions for the custom message types used in the SAC system. These are described in Section 3.2.
sac_config	Not a ROS package, but a project used for setting up ROS, Gazebo, and the SAC system.
scorbot_config	Holds the package generated by the MoveIt! Setup Assistant for the scorbot and MoveIt!.
andreas_arm_config	Holds the package generated by the MoveIt! Setup Assistant for use of the Andreas arm and MoveIt!.

Table 3.2: The custom packages used in the SAC system. the scorbot_config, andreas_arm_config, and sac_config packages are not currently in use in the SAC system as they are only necessary for MoveIt!.

[49, 50, 51, 52, 53, 54, 55, 56, 57, 71]

- ROS Computational Graph Messages – includes messages used internally in the ROS system [60].

Messages are published on topics that can be received by any node. The SAC system contains several custom ROS message types that are included in the sac_msgs project for both node and service messages [54], shown in Table 3.3.

Name	Type	Include File	Purpose
HandPos	Message	sac_msgs\HandPos.h	Tells the hand the width it should be at.
Path	Message	sac_msgs\Path.h	Defines a line for the end-effector of the arm to move along.
Target	Message	sac_msgs\Target.h	Defines a goal for the end-effector to move to.
Encoder	Service	sac_msgs\Encoder.h	Determines if a joint switch is hit.
MotorPosition	Service	sac_msgs\MotorPosition.h	Stepper ticks for the motor to move.
MotorsComplete	Service	sac_msgs\MotorsComplete.h	Whether all motors have completed movements.
MotorsOffset	Service	sac_msgs\MotorsOffset.h	A home offset to set the motors to.
MotorSpeed	Service	sac_msgs\MotorSpeed.h	The speed for a motor to move at 1 through 9 then 0.

Table 3.3: The messages provided by the sac_msgs project [54]. [61]

3.3 Structure

The control system is used to control either the physical or simulated robot, or both simultaneously. The control structure shown in Figure 3.1 is divided into three sections: controllers, translators, and drivers. Controllers provide overall instructions to the system. Translators convert from the controller's global goals to the joint space goals of the arm. Drivers will send instructions to the hardware and simulator.

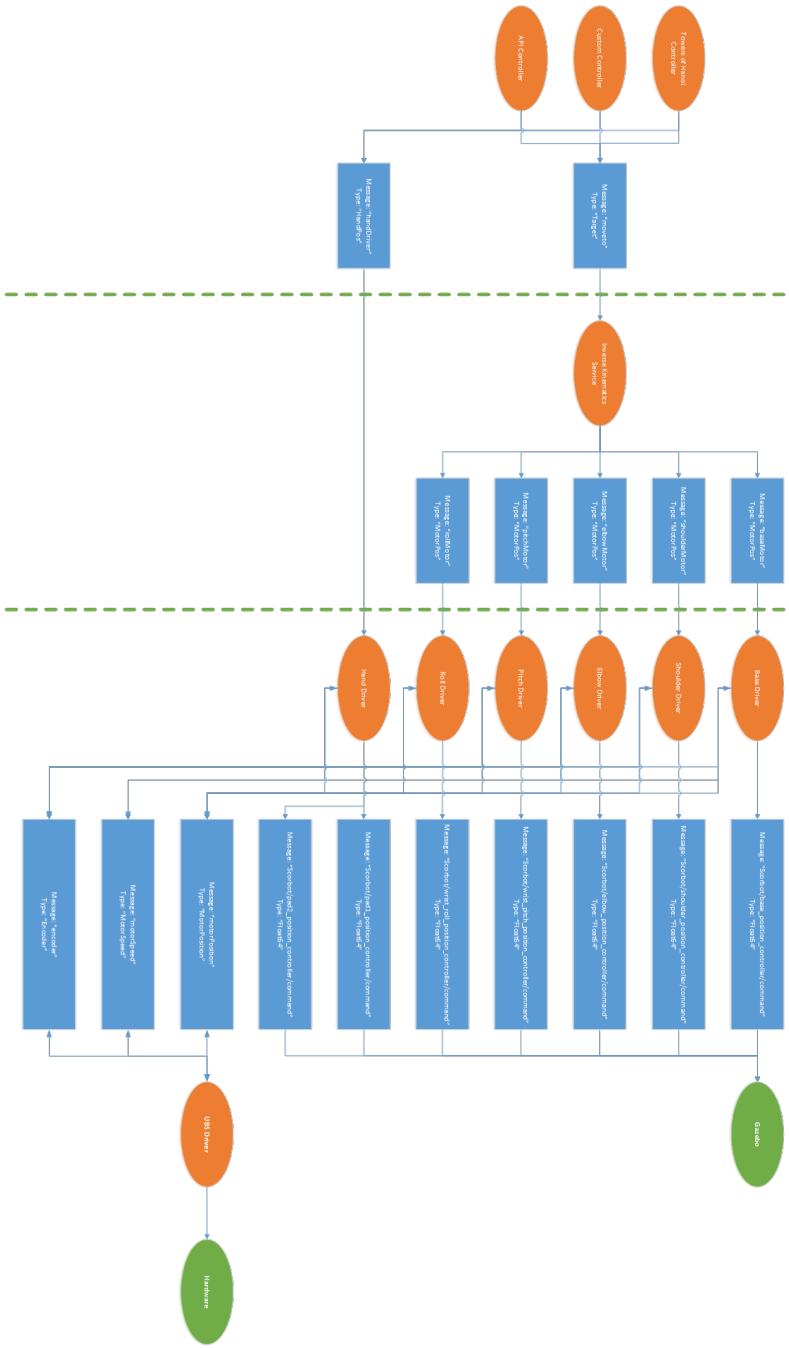

Figure 3.1: The topology of the implemented ROS system is detailed in the above figure. The orange nodes represent the individual ROS nodes that make up the system, green nodes indicate external items, and blue indicate the ROS messages passed with the given message topic and type. The first section to the left is the controllers, in the middle are the translators, and on the right are the drivers. These sections are detailed in Figures 3.2, 3.3, 3.6 respectively.

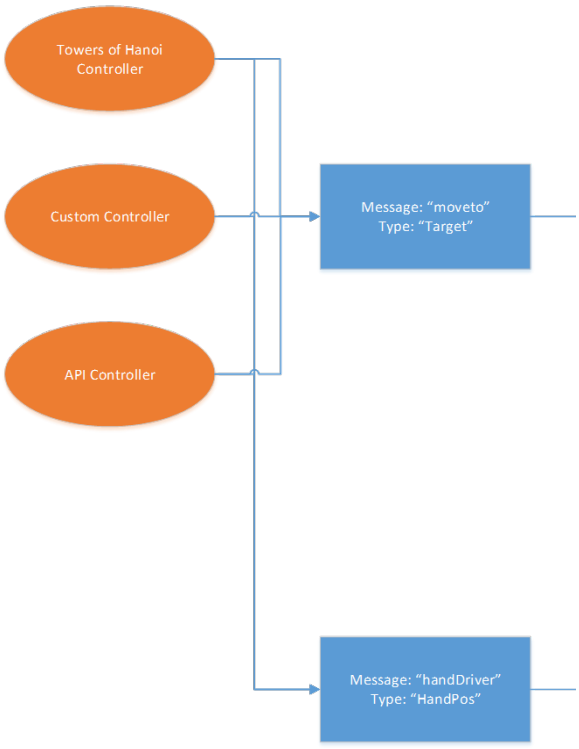

Figure 3.2: Controllers are used to send overall goals for the arm.

3.3.1 Controllers

The controllers of the system provide overall instructions to the system. The controllers are detailed in Figure 3.2.

3.3.1.1 Towers of Hanoi

The Towers of Hanoi controller provides a demo of the robot's movements in a static scenario This controller is written in C++. It publishes "moveto" messages of type "Target" with end-effector positions, and "handDriver" messages of type "HandPos" with hand widths.

3.3.1.2 API Controller

The API controller takes in input from an Internet connection and publishes commands to the system. This controller is written in Python. The node uses get requests and parses the URL for each parameter of the arm movement. The URL formatting is as ”ip:8080/x/y/z/roll/pitch/handWidth/time” where ”ip” is the IP address of the computer running the node, ”x”, ”y”, and ”z” are the Cartesian coordinates of the end-effector, ”pitch” and ”roll” are the angle parameters of the end-effector, ”handWidth” is the opening width of the hand, and ”time” is the time taken for these moves. The code for this node is a common standard form for a Python Server and is similar to the following sources [62, 63, 64]. It publishes ”moveto” messages of type ”Target” with end-effector positions, and ”handDriver” messages of type ”HandPos” with hand widths. A test program has been written as a shell script to communicate with the API over the Internet and perform the Towers of Hanoi solution. This program is placed in the examples folder of the sac_setup project [42].

3.3.1.3 Custom Controller

The Custom controller is a template for use by students when building their own nodes. It may be copied and easily modified with little knowledge of ROS or the rest of the SAC system.

3.3.2 Translators

Translators convert incoming Cartesian goals to joint space goals for the motors. The translators are detailed in Figure 3.3.

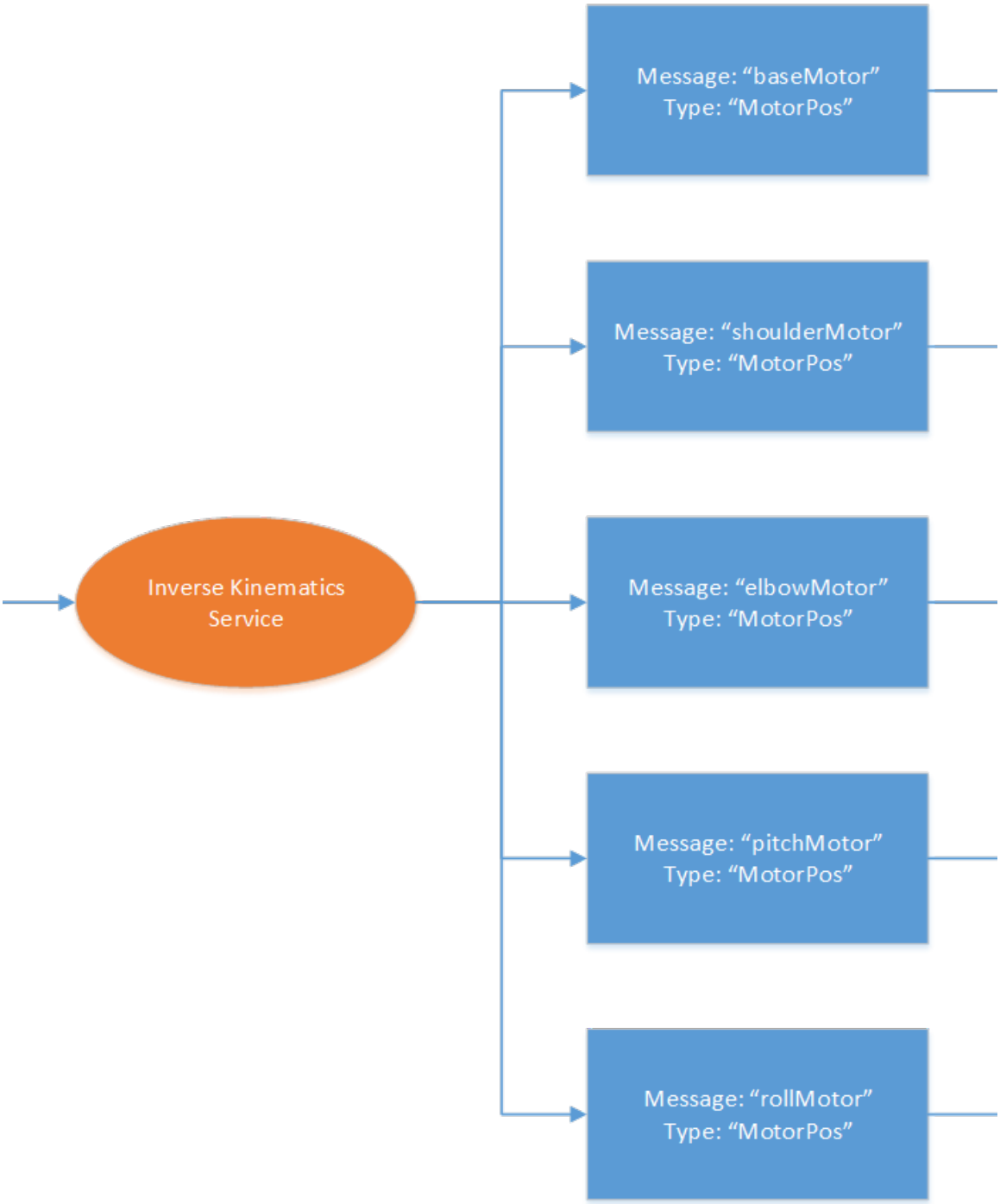

Figure 3.3: Translators are used to translate the inputs from the controllers to the joints.

3.3.2.1 Scorbot IK

This translator performs the inverse kinematics and goal checking for the Scorbot er-III. It takes in ”moveto” messages of type ”Target” with end-effector positions, and publishes ”baseMotor”, ”shoulderMotor”, ”elbowMotor”, ”pitchMotor”, and ”rollMotor” messages of type ”MotorPos” with radian angles for each motor. Inverse kinematics equations can be derived one of two ways. First is geometrically [65]. This method looks at the joints of the robot and uses trigonometric identities to calculate the angles of the robot [65]. The second method is through the Denavit-Hartenberg (D-H) method. This method uses tables and matrices generated by the robot configuration to produce equations [66]. This project uses the Geometric approach, outlined by [65], as it is more straight forward and self-explanatory. For all equations and modeling, the ROS standard units will be used for measurements as follows:

- Lengths: Meters (m) [67]
- Angles: Radians [67]
- Time: Seconds (s) [67]
- Weights: Kilograms (kg) [67]

The following equations are used for each motor translation:
For the given arm lengths (meters):

- $d_0 = 0.360$ [68]
- $d_1 = 0.030$ [68]
- $d_2 = 0.220$ [68]

- $d_3 = 0.220$ [68]
- $d_4 = 0.140$ [68]

For the given limits (radians):

- $-2.7053 < \theta_B < 2.7053$
- $-0.6109 < \theta_S < 2.2689$
- $-2.2689 < \theta_E < 2.2689$
- $-2.2689 < \theta_P < 2.2689$

The base angle (θ_B) for the robot, illustrated in figure 3.4, is determined by

$$\theta_B = atan2(y, x) \tag{3.1}$$

Where x and y are the given end-effector goals for their respective axis.

The projected height (d_{4z}) and radius (d_{4r}) of the d_4 link can be found using the input θ_{PITCH} by the equations

$$d_{4z} = sin(\theta_{PITCH}) \cdot d_4 \tag{3.2}$$

and

$$d_{4r} = cos(\theta_{PITCH}) \cdot d_4 \tag{3.3}$$

It is now beneficial to determine the total length from the center of the robot to its end-effector, the total radius (r). This can be done by using the inputs x and y.

$$r = \sqrt{x^2 + y^2} \tag{3.4}$$

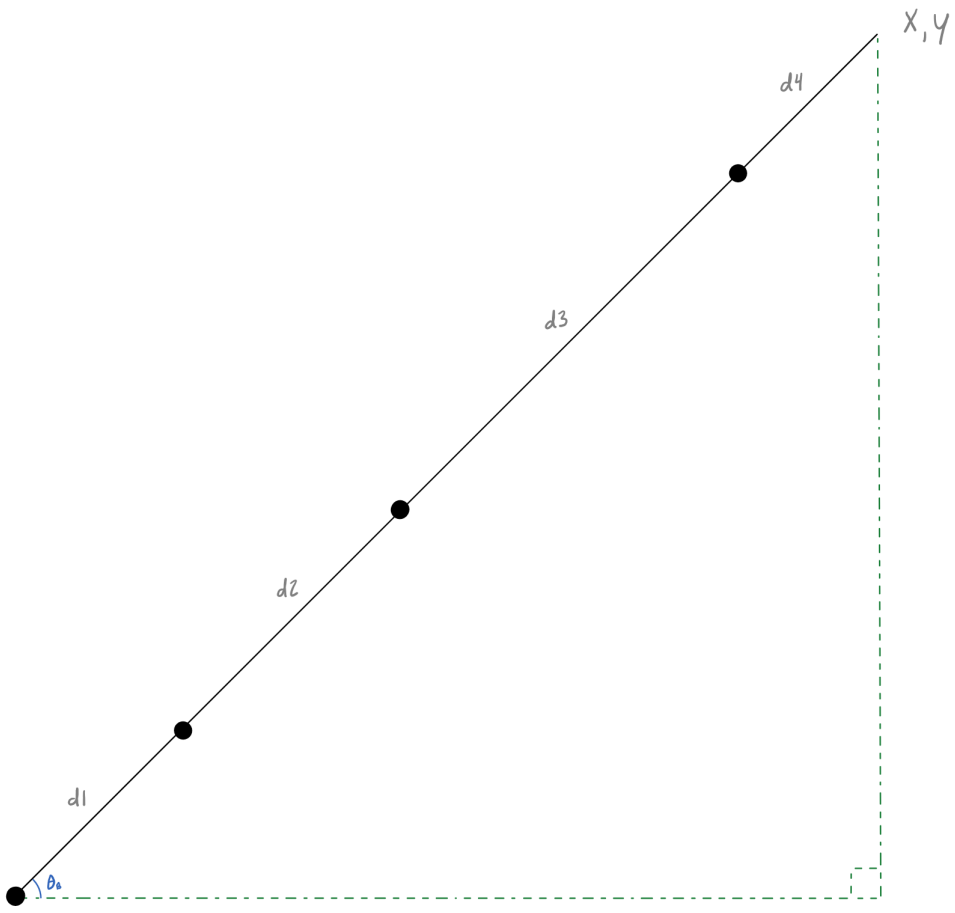

Figure 3.4: A side view of the arm with the inverse kinematics variables. Black solid lines indicate the links of the robot in the primary position. Black dashed lines indicate the alternate positions of the robot links. Large black dots indicate the joints of the robot. Blue angles, dashed lines, and labels indicate the joint angles of the robot. Green dashed lines, angles and labels indicate the intermediate variables used in the calculation of the joint angles. Gray labels indicate the known values of the end-effector that are passed in and for the lengths of the arm links.

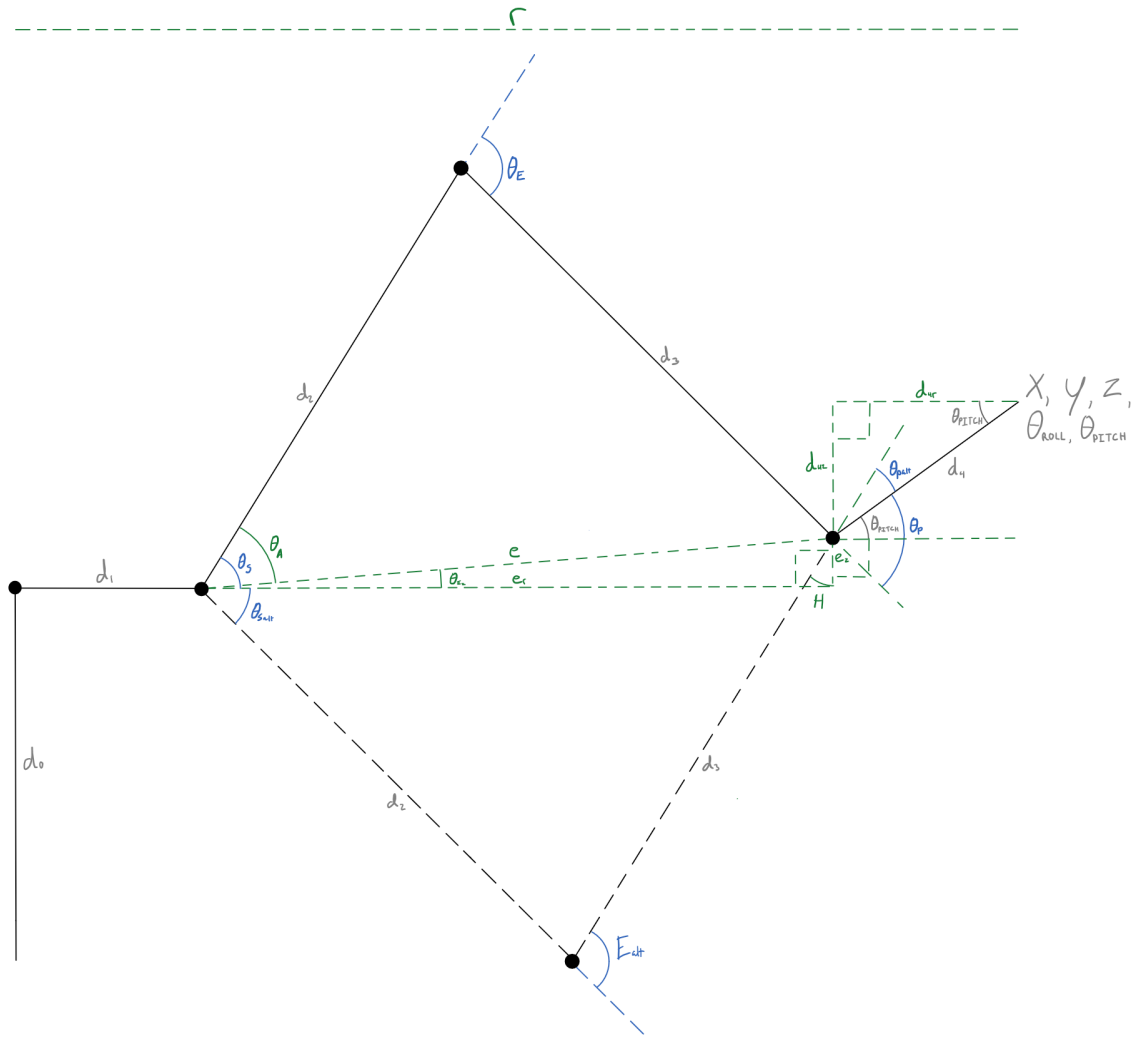

Figure 3.5: A top view of the arm with the inverse kinematics variables. Black solid lines indicate the links of the robot in the primary position. Black dashed lines and labels indicate the alternate positions of the robot links. Large black dots indicate the joints of the robot. Blue angles, dashed lines, and labels indicate the joint angles of the robot. Green dashed lines, angles and labels indicate the intermediate variables used in the calculation of the joint angles. Gray labels indicate the known values of the end-effector that are passed in and for the lengths of the arm links.

Using the results of the Equations 3.2 and 3.3, the height (e_z) and radius (e_r) change from the shoulder joint (θ_S) to the wrist pitch joint (θ_P) opposite the elbow joint (θ_E) can now be derived as

$$e_z = z + d_{4z} - d_0 \tag{3.5}$$

and

$$e_r = r - d_1 - d_{4r} \tag{3.6}$$

Using the values produced by Equations 3.5 and 3.6, the total length from the shoulder (θ_S) to the wrist pitch joints (θ_P) found using

$$e = \sqrt{e_r^2 + e_z^2} \tag{3.7}$$

The elbow joint (θ_E) can now be determined. This equation comes from [65].

$$\theta_E = atan2(\sqrt{1 - \frac{e_r^2 + e_z^2 - d_2^2 - d_3^2}{2 \cdot d_2 \cdot d_3}^2}, \frac{e_r^2 + e_z^2 - d_2^2 - d_3^2}{2 \cdot d_2 \cdot d_3}) \tag{3.8}$$

The angle θ_{E_z} between the e_r and e measurements can be found using the law of cosines as

$$\theta_{E_z} = acos(\frac{e_z^2 - e^2 - e_r^2}{-2 \cdot e \cdot e_r}) \tag{3.9}$$

Using the law of cosines again the angle θ_A between the e measurement and the d_4 link can be found by

$$\theta_A = acos(\frac{d_3^2 - d_2^2 - e^2}{-2 \cdot d_2 \cdot e}) \tag{3.10}$$

Using the Equations 3.9 and 3.10, the shoulder joint (θ_S) can be found, as shown by

$$\theta_S = \theta_A - \theta_{E_z} \tag{3.11}$$

The wrist pitch joint can now be found using the known angles around the joint as shown by

$$\theta_P = \theta_{PITCH} - (\theta_E - \theta_S) \tag{3.12}$$

The wrist roll angle (θ_R) is known from the given user inputs as is given by the equation

$$\theta_R = \theta_{ROLL} \tag{3.13}$$

3.3.3 Drivers

The drivers for the system take in messages from all other nodes and send commands to the hardware of the system. The drivers are detailed in Figure 3.6.

3.3.3.1 Base, Shoulder, Elbow, Pitch, and Roll Drivers

These drivers take inputs, check them, and publish them to various messages to control their respective motors. They all take in "{joint}Motor" messages, where {joint} is the name of the joint, of type "MotorPos" and publishes "scorbot/{joint}_position_controller/command" messages, with "wrist_" added in front of "[joint]" for the wrist joints, of type "Float64", "motorPosition" messages of type "MotorPosition", "motorSpeed" messages of type "MotorSpeed", and "encoder" messages of type "Encoder".

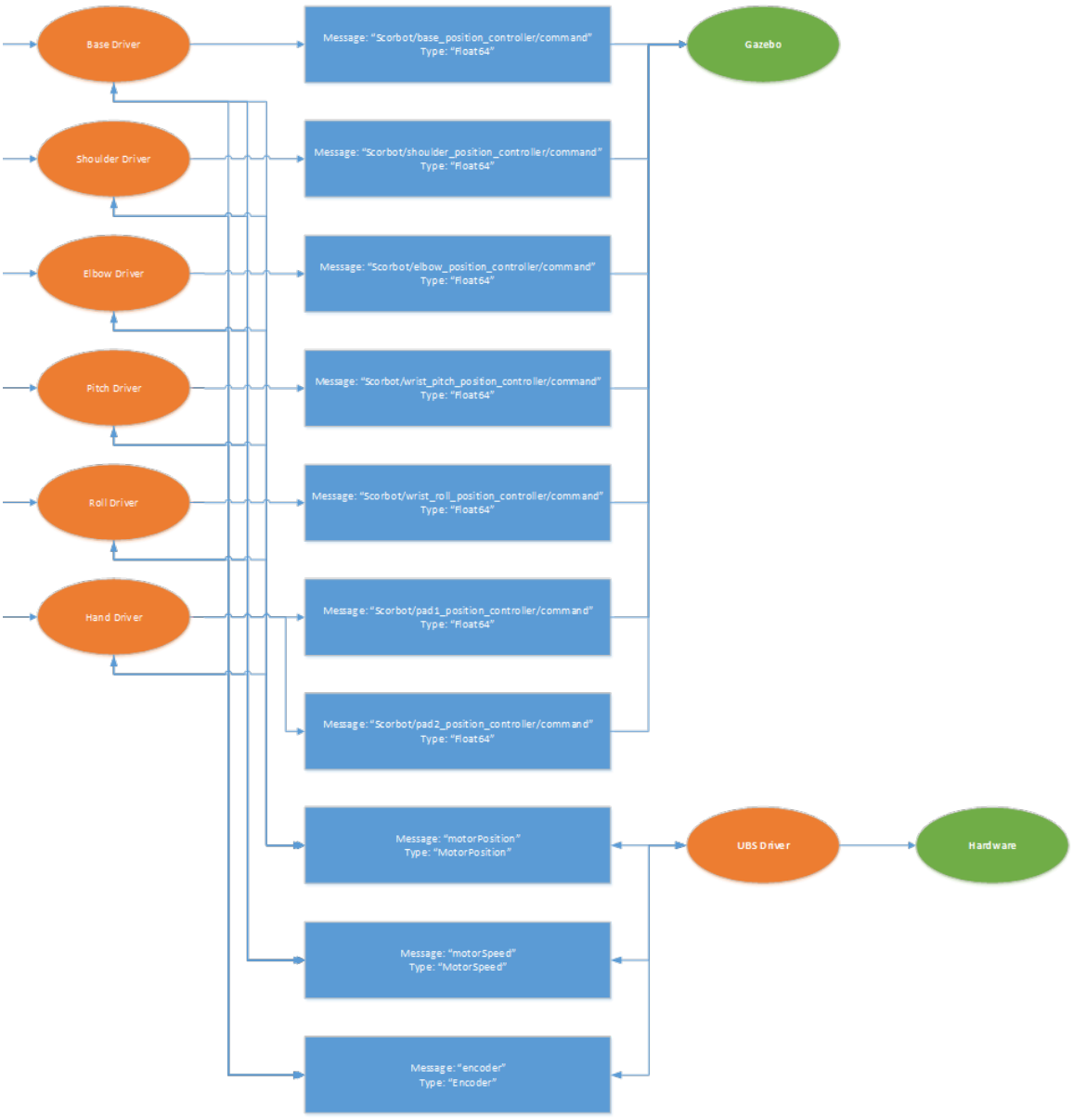

Figure 3.6: Drivers are used to do low level hardware interfacing.

Topic	Type	In/Out
base_driver		
baseMotor	MotorPos	In
scorbot/base_position_controller/command	Float64	Out
motorPosition	MotorPosition	Out
motorSpeed	MotorSpeed	Out
encoder	Encoder	Out
shoulder_driver		
shoulderMotor	MotorPos	In
scorbot/shoulder_position_controller/command	Float64	Out
motorPosition	MotorPosition	Out
motorSpeed	MotorSpeed	Out
encoder	Encoder	Out
elbow_driver		
elbowMotor	MotorPos	In
scorbot/elbow_position_controller/command	Float64	Out
motorPosition	MotorPosition	Out
motorSpeed	MotorSpeed	Out
encoder	Encoder	Out
pitch_driver		
pitchMotor	MotorPos	In
scorbot/wrist_pitch_position_controller/command	Float64	Out
motorPosition	MotorPosition	Out
motorSpeed	MotorSpeed	Out
encoder	Encoder	Out
roll_driver		
rollMotor	MotorPos	In
scorbot/wrist_roll_position_controller/command	Float64	Out
motorPosition	MotorPosition	Out
motorSpeed	MotorSpeed	Out
encoder	Encoder	Out

Table 3.4: A summary of the inputs and outputs of the motor drivers.

3.3.3.2 USB Driver

This driver is a series of services that communicate with the serial interface of the robotic arm through a USB adapter. This node combines several services into one to allow for communication with the USB without interruption, as ROS queues

each request and runs them serially. This node takes in ”motorSpeed” messages of type ”MotorSpeed”, ”motorPosition” messages to type ”MotorPosition”, ”motorsOffset” messages of type ”MotorsOffset”, ”motorsComplete” messages of type ”MotorsComplete”, and ”encoder” messages of type ”Encoder”.

3.4 Simulation

For testing the software a simulator is used to show the output from SAC system, This Gazebo simulator works on Windows OSX, and Linux [69]. This simulator is free and open source. The model of the Scorbot used for the project is modified from [70].

3.5 Hardware

Due to time limitations, this project has choden to use the SCORBOT-ER III as its robotic arm. The Scorbot is a simple 5 DoF, available to the school currently.

3.6 Source Control

The system has been placed on GitHub for source control and documentation. A ROS workspace contains absolute file paths and does not transfer well from system to system. The project has been published as individual packages and a project for installation these can be found at https://github.com/greenpro/ [50, 49, 71, 51, 52, 53, 54, 55, 56, 42].

3.7 Documentation

The project is documented directly in readme files that are in every folder and project. These documentation files detail the files, subfolders, removable files, and any notes for the immediate folder.

3.8 Installation

The system can be installed on Ubuntu 16.04 by running installing git using the apt-get install command. The user can then clone the SAC Setup project and run the sac_setup.sh script [42]. To build and run the system, the user may run "./build.sh" from the " /sac/" directory, then "roslaunch sac_launch {launch file}" where "{launch file}" is the first word of the controller to be used followed by a ".launch" (e.g. "roslaunch sac_launch towers.launch").

3.9 Educational Uses

A robotic arm can be used to teach many difficult concepts in robotics and other classes as shown in previously mentioned works. The initial use of this particular arm will be specific to a graduate robotics class, though future uses could go far beyond just one class. The initial concepts this robot is intended to teach include: robot construction and design, kinematics, inverse kinematics, Jacobians, and robot control frameworks. This list may be expanded later on to include more concepts and areas in the future.

3.10 Bill of Materials

The bill of materials was modified by the proposal committee to use a Raspberry Pi 3 and The SCORBOT-ER III available in the lab. Thus, no hardware or software acquisitions were made in the completion of this project.

3.11 Tasks and Milestones

Milestones	Task Group	Projected Hours	Actual Hours
1	Documentation	40	25
1	Research and BOM	220	268
2	ROS Setup	4	14
2	Raspberry Pi Setup	12	18
3	Boot Sequence	12	-
4	Simulator	20	72
5	Motor Drivers	14	13
5	Encoder Drivers	10	11
6	Hand Drivers	10	9
7	Menu Service	8	-
7	Display Driver	4	13
8	Custom Controller	4	2
8	Inverse Kinematics Service	18	130
9	Jacobian Service	18	25
10	Towers of Hanoi Controller	18	60
11	API Controller	22	6
-	Project Meetings	-	16
Total Hours		434	682

Table 3.5: The above table shows the list of task groups for each module. The modules and tasks are in the order of completion for the project. A full list of tasks can be found in Appendix

The Table 3.5 shows the list of milestones shown as project modules in the first column. Under each milestone is a list of task groups.

The deliverables for the project include a software stack, documentation, and

simulation of the a robot arm. The system is set up for use in a classroom environment and is ready for labs.

Parts of the project have been delayed due to others taking longer than they were expected. This is the case in the use of MoveIt!, included in the Inverse Kinematics and Jacobian Services task groups. MoveIt! had many issues in its setup, as well as, a general misunderstanding of the software's capabilities, by the author, in its the current version. The second part of the project that took a large amount of time is the difficulty of grasping items in the Gazebo simulator; this time is factored into the Towers of Hanoi Controller. Much of the research spent at the start of the project comes from time spent looking at other projects and the process of settling on the current project. Due to time issues, the Raspberry Pi has been replaced by a PC running Ubuntu 16.04. For this reason the Boot Sequence and Menu Service tasks were not completed as they are only necessary for the Raspberry Pi. The Jacobian node has been taken out as it was not needed, as the SCORBOT robots are not good at drawing and moving with precise speeds [31, 61].

Chapter 4

Results

The SAC system consists of eleven projects, documented by thirty readme files. Ten packages are in the system. They build eleven nodes that publish on seventeen topics with nine custom message types and a standard message type. Six external packages are used in the system for integration and fixes.

4.1 ROS Nodes

The system is easy to run by simply using one of the provided launch files in the SAC launch project. The system functionality has been demonstrated through several controllers. These controllers display pre-programmed movements and real-time integration.

The Inverse Kinematics for the robot were worked out by hand using the geometric method. This was significantly slowed down due to automated tools, such as MoveIt! and its plug-in IKfast not working as expected.

The Towers of Hanoi controller has successfully completed moving the tower of blocks from one position to another. The sequence takes approximately 7.5

minutes to run and completes a three block tower in seven moves. This node demonstrates a C++ node for the system. Due to the issues with gripping objects, for this controller repeatability is low and may only last for one run before the tower collapses.

Gazebo causes a few issues with this node. The first is the ability to grasp objects. This issue has been fixed through the addition of the gazebo_grasp_plugin by [48]. A second issue is with the Gazebo simulator's ability to simulate the physics of an object. Objects can sometimes sink into the floor or other objects. This issue was not as easily overcome as the gripper and remains to be an issue with the simulation. Time constraints did not allow for these issues to be investigated further.

The API controller for the system allows an external piece of code to control the program through an Internet connection. The API controller is an example of a python node. A shell script has been written to test the node by running Linux wget commands on the API.

4.2 Hardware

A Raspberry Pi was not used in the system as a laptop computer was readily available and allows for a much more flexible system. This also allowed time to be spent elsewhere as time was limited for this project.

4.3 Installation

The project can be easily installed through the scripts provided in the SAC setup project. Installing the system, from Ubuntu 16.04 download, to a fully running

system, takes approximately an hour. The installer downloads and installs all necessary components for the system with the exception of Git, which must be downloaded separately, since the project resides on Github.

4.4 Documentation

The project has been documented through the use of Github-formatted readme files. These provide all the necessary information for running, editing, and adding to the system.

Chapter 5

Conclusion

The SAC system is a software control stack built on ROS for a robotic arm built on ROS for use in a classroom environment. Students may implement, modify and test various parts of the system for classes and projects. The functionality of the project is implemented in the control of the 3D simulation of the 5 DoF SCORBOT-ER III robotic arm.

5.1 Future Work

Future work for the project will consist first of completing the integration of MoveIt! into the SAC system. This will allow objects to be picked up more consistently. Most of the issues with picking things up comes from the fact that Gazebo is not fully developed and is still missing some features [72]. Other fixes will help to make picking up and putting down objects more consistent. Node testing should also be added to the project to ensure correctness [73]. The SAC system should be integrated with the SCORBOT-ER III hardware in the Future, and possibly move to the arm by Andreas Hölldorfer [25, 26].

Appendix A

Requirements

A.1 Software

Software requirements for the system. The italicized items were not completed.

A.1.1 Raspberry Pi Software

Controller computer requirements for the system. The italicized items were not completed.

1. *The Raspberry Pi should run the Raspian Operating System.*
2. *The Raspberry Pi will run ROS for the robot control.*
3. The ROS system should contain a node for controlling the motors.
4. The ROS system should contain a node for controlling the encoders.
5. The ROS system should contain a node for controlling the hand servos.
6. *The ROS system should contain a node for controlling the display text and managing the menu system.*

7. The ROS system should contain an inverse kinematics service node.
8. *The ROS system should contain a Jacobian service node.*
9. The ROS system should contain a custom controller node.
10. The ROS system should contain a Towers of Hanoi Controller node.
11. The ROS system should contain an API controller node.

A.1.2 Simulation and Remote Control Software

Simulation and remote control software requirements for the system.

1. The remote computer should run the simulator for the robot.
2. The remote computer should be able to run all of the Raspberry Pi nodes for use with the simulator or remote operation.
3. The remote computer should be able to run any of the control nodes for remote programming and testing of the system.
4. The remote system should be able to run a web browser to interact with the API node of the robot.

A.2 Installation

System installation requirements for the system.

1. The user should be able to install the robot software by copying the project files from the repository to the project on their own system.

A.3 Running

Requirements for running the system the system. The italicized items were not completed.

1. The project should be able to be run through a ROS launch file that will launch all of the nodes on the system.
2. *The system on the Raspberry Pi should startup automatically once Raspian has started as a background task.*

A.4 Modification

Modification requirements for the system. The italicized items were not completed.

1. Students should be able to modify the system by removing and rewriting a node.
2. Students should be able to write the code for the custom controller to modify the arm for other uses.

Bibliography

[1] M. F. Silva, B. Curto, and V. Moreno, "A robot in the classroom," in *Proceedings of the 3rd International Conference on Technological Ecosystems for Enhancing Multiculturality*, ser. TEEM '15. New York, NY, USA: ACM, 2015, pp. 197–201. [Online]. Available: http://doi.acm.org.ezproxy.southern.edu/10.1145/2808580.2808610 1, 2.1

[2] S. P. N. Singh, H. Kurniawati, K. S. Naveh, J. Song, and T. Zastrow, "CHARM: A platform for algorithmic robotics education amp; research," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 2602–2607. 1, 2.1

[3] J. Shin, A. Rusakov, and B. Meyer, "Concurrent software engineering and robotics education," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 370–379. [Online]. Available: http://dl.acm.org.ezproxy.southern.edu/citation.cfm?id=2819009.2819068 1, 2.1

[4] F. Cuellar, D. Arroyo, E. Onchi, and C. Penaloza, "IREP: An interactive robotics education program for undergraduate students," in *Robotics Symposium and Competition (LARS/LARC), 2013 Latin American*, Oct 2013, pp. 153–158. 1, 2.1

[5] P. Bouchier. (2015, April) A gentle introduction to ROS (and related technologies). [Online]. Available: https://dprgblog.files.wordpress.com/2015/04/2015marchrostalk-1.pdf 1.1, 2.3, 3.1

[6] T. L. Dunn and A. Wardhani, "A 3D robot simulation for education," in *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, ser. GRAPHITE '03. New York, NY, USA: ACM, 2003, pp. 277–278. [Online]. Available: http://doi.acm.org.ezproxy.southern.edu/10.1145/604471.604535 2.1

[7] T. Sapounidis, S. Demetriadis, and I. Stamelos, "Evaluating children performance with graphical and tangible robot programming tools," *Personal Ubiquitous Comput.*, vol. 19, no. 1, pp. 225–237, Jan. 2015. [Online]. Available: http://dx.doi.org.ezproxy.southern.edu/10.1007/s00779-014-0774-3 2.1

[8] A. Saad and R. M. Kroutil, "Hands-on learning of programming concepts using robotics for middle and high school students," in *Proceedings of the 50th Annual Southeast Regional Conference*, ser. ACM-SE '12. New York, NY, USA: ACM, 2012, pp. 361–362. [Online]. Available: http://doi.acm.org.ezproxy.southern.edu/10.1145/2184512.2184605 2.1

[9] M. Rubenstein, B. Cimino, R. Nagpal, and J. Werfel, "AERobot: An affordable one-robot-per-student system for early robotics education," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 6107–6113. 2.1

[10] E. R. Doering, "Electronics lab bench in a laptop: using electronics workbench #174; to enhance learning in an introductory circuits course," in *Frontiers in*

Education Conference, 1997. 27th Annual Conference. Teaching and Learning in an Era of Change. Proceedings., vol. 1, Nov 1997, pp. 18–21 vol.1. 2.1

[11] LEGO. (2016) Learn to program - it's easy. [Online]. Available: http://www.lego.com/en-us/mindstorms/learn-to-program 2.1, 2.3

[12] FIRST. What is FIRST LEGO league. [Online]. Available: http://www.firstlegoleague.org/about-fll 2.1

[13] W.-P. Xu, W. Li, and L.-G. Liu, "Skeleton-sectional structural analysis for 3D printing," *Journal of Computer Science and Technology*, vol. 31, no. 3, pp. 439–449, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11390-016-1638-2 2.2

[14] M. Mukhtar, E. Akyrek, T. Kalganova, and N. Lesne, "Control of 3D printed ambidextrous robot hand actuated by pneumatic artificial muscles," in *SAI Intelligent Systems Conference (IntelliSys), 2015*, Nov 2015, pp. 290–300. 2.2

[15] M. Lapeyre, P. Rouanet, J. Grizou, S. N'Guyen, A. L. Falher, F. Depraetre, and P. Y. Oudeyer, "Poppy: Open source 3D printed robot for experiments in developmental robotics," in *4th International Conference on Development and Learning and on Epigenetic Robotics*, Oct 2014, pp. 173–174. 2.2

[16] M. Lapeyre, S. N'Guyen, A. L. Falher, and P. Y. Oudeyer, "Rapid morphological exploration with the Poppy humanoid platform," in *2014 IEEE-RAS International Conference on Humanoid Robots*, Nov 2014, pp. 959–966. 2.2

[17] M. Lapeyre, P. Rouanet, and P. Y. Oudeyer, "Poppy humanoid platform: Experimental evaluation of the role of a bio-inspired thigh shape," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Oct 2013, pp. 376–383. 2.2

[18] h. oehm. (2013, March) OpenScad micro servo robot arm. [Online]. Available: http://www.thingiverse.com/thing:65081 2.2

[19] C. Franciscone. (2015, September) EEZYbotArm. [Online]. Available: http://www.thingiverse.com/thing:1015238 2.2

[20] C. Arnø. (2014, May) 4 axis robotic arm. [Online]. Available: http://hacknorway.com/wordpress/4-axis-robotic-arm/ 2.2

[21] angelLM. (2016) Thor. [Online]. Available: https://hackaday.io/project/12989-thor 2.2

[22] Zortrax. (2016) Get your free 3D files for the robotic arm. [Online]. Available: https://zortrax.com/free-robotic-arm-files/ 2.2

[23] BCN3D. (2016, July) BCN3D MOVEO a fully open source 3D printed robot arm. [Online]. Available: https://www.bcn3dtechnologies.com/en/bcn3d-moveo-the-future-of-learning/ 2.2

[24] ——. (2016, October) BCN3D-Moveo: Open source 3D printed robotic arm for educational purposes. [Online]. Available: https://github.com/BCN3D/BCN3D-Moveo 2.2

[25] A. Hölldorfer. (2016, April) Chaozlabs. [Online]. Available: http://chaozlabs.blogspot.de/ 2.2, 5.1

[26] ——. (2016, February) 3D printable robot arm. [Online]. Available: https://github.com/4ndreas/BetaBots-Robot-Arm-Project 2.2, 5.1

[27] S. Robotics. (2016, March) Most advanced robotics simulation software overview. [Online]. Available: https://www.smashingrobotics.com/most-advanced-and-used-robotics-simulation-software/ 2.3

[28] Vex. (2016) Why VEX IQ. [Online]. Available: http://www.vexrobotics.com/vexiq/why-vexiq 2.3

[29] J. F. M. Lee and J. A. Buitrago, "Map generation and localization for a LEGO NXT robot," in *Automatic Control (CCAC), 2015 IEEE 2nd Colombian Conference on*, Oct 2015, pp. 1–5. 2.3

[30] M. Pinto, A. P. Moreira, and A. Matos, "Localization of mobile robots using an Extended Kalman Filter in a LEGO NXT," *IEEE Transactions on Education*, vol. 55, no. 1, pp. 135–144, Feb 2012. 2.3

[31] L. Cole, A. F. Nagy-Sochacki, and J. Symonds. (2007, October) Drawing using the SCORBOT-ER VII manipulator arm. [Online]. Available: https://www.lukecole.name/doc/reports/drawing_using_the_scorbot_manipulator_arm.pdf 2.3, 3.11

[32] J. M. G. de Gabriel, A. Mandow, J. Fernandez-Lozano, and A. J. Garcia-Cerezo, "Using LEGO NXT mobile robots with LabVIEW for undergraduate courses on mechatronics," vol. 54, no. 1, pp. 41–47, Feb 2011. 2.3

[33] C. Ltd. (2016) Buy Webots. [Online]. Available: http://www.cyberbotics.com/buy 2.3

[34] O. S. R. Foundation. (2014) Gazebo tutorials. [Online]. Available: http://gazebosim.org/tutorials?cat=install 2.3

[35] DHood. (2016, May) Debian install of ROS Kinetic. [Online]. Available: http://wiki.ros.org/kinetic/Installation/Debian 2.3

[36] D. Rivas, M. Alvarez, P. Velasco, J. Mamarandi, J. L. Carrillo-Medina, V. Bautista, O. Galarza, P. Reyes, M. Erazo, M. Prez, and M. Huerta, "BRACON:

Control system for a robotic arm with 6 degrees of freedom for education systems," in *Automation, Robotics and Applications (ICARA), 2015 6th International Conference on*, Feb 2015, pp. 358–363. 2.3

[37] M. Ferguson, N. Webb, and T. Strzalkowski, "Nelson: A low-cost social robot for research and education," in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 225–230. [Online]. Available: http://doi.acm.org.ezproxy.southern.edu/10.1145/1953163.1953230 2.3

[38] Y. Hold-Geoffroy, M. A. Gardner, C. Gagn, M. Latulippe, and P. Gigure, "ros4mat: A Matlab programming interface for remote operations of ROS-based robotic devices in an educational context," in *Computer and Robot Vision (CRV), 2013 International Conference on*, May 2013, pp. 242–248. 2.3

[39] O. S. R. Foundation. (2017, April) ROS tutorials. [Online]. Available: http://wiki.ros.org/ROS/Tutorials 3.1

[40] D. Kurzaj. (2016, August) Messages. [Online]. Available: http://wiki.ros.org/Messages 3.1

[41] K. Conley. (2012, February) Services. [Online]. Available: http://wiki.ros.org/Services 3.1

[42] C. Christensen. (2017, April) Southern Arm Control Setup. [Online]. Available: https://github.com/greenpro/sac_setup 3.1, 3.3.1.2, 3.6, 3.8

[43] Installing gazebo_ros_pkgs. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros 3.1

[44] (2017, March) osrf/gazebo8_ros_pkgs-release. Open-Source Robotics Foundation. [Online]. Available: https://github.com/osrf/gazebo8_ros_pkgs-release 3.1

[45] W. Meeussen, A. Rodriguez, B. Magyar, D. Coleman, and E. Fernandez. (2017, March) ros_control. [Online]. Available: http://wiki.ros.org/ros_control 3.1

[46] ——. (2013, June) ros_controllers. [Online]. Available: http://wiki.ros.org/ros_controllers 3.1

[47] C. S. Sucan, Ioan, D. Coleman, M. Ferguson, M. Görner, R. Haschke, I. McMahon, and I. I. Y. Saito. (2016, August) moveit. [Online]. Available: http://wiki.ros.org/moveit 3.1

[48] J. Buehlser. (2016, March) The Gazebo grasp fix plugin. [Online]. Available: https://github.com/jenniferBuehler/gazebo-pkgs/wiki/The-Gazebo-grasp-fix-plugin 3.1, 4.1

[49] C. Christensen. (2017, April) Southern Arm Control Controllers. [Online]. Available: https://github.com/greenpro/sac_controllers 3.2, 3.6

[50] ——. (2017, April) Southern Arm Control Drivers. [Online]. Available: https://github.com/greenpro/sac_drivers 3.2, 3.6

[51] ——. (2017, April) Southern Arm Control Gazebo. [Online]. Available: https://github.com/greenpro/sac_gazebo 3.2, 3.6

[52] ——. (2017, April) Southern Arm Control Launch. [Online]. Available: https://github.com/greenpro/sac_launch 3.2, 3.6

[53] ——. (2017, April) Southern Arm Control Translators. [Online]. Available: https://github.com/greenpro/sac_translators 3.2, 3.6

[54] ——. (2017, April) Southern Arm Control Messages. [Online]. Available: https://github.com/greenpro/sac_msgs 3.2, 3.2, 3.3, 3.6

[55] ——. (2017, April) Scorbot Config. [Online]. Available: https://github.com/greenpro/scorbot_config 3.2, 3.6

[56] ——. (2017, April) Andreas Arm Config. [Online]. Available: https://github.com/greenpro/andreas_arm_config 3.2, 3.6

[57] ——. (2017, April) Southern Arm Control Config. [Online]. Available: https://github.com/greenpro/sac_config 3.2

[58] M. Quigley, K. Conley, J. Leibs, and T. Foote. (2017, March) std_msgs. [Online]. Available: http://wiki.ros.org/std_msgs 3.2

[59] T. Foote. (2014, April) common_msgs. [Online]. Available: http://wiki.ros.org/common_msgs 3.2

[60] k. Conley and D. Thomas. (2015, May) rosgraph_msgs. [Online]. Available: http://wiki.ros.org/rosgraph_msgs?distro=kinetic 3.2

[61] E. R. . Limited. (1999, December) SCORBOT-ER III user's manual. [Online]. Available: http://www.theoldrobots.com/book45/ER3-Manual.pdf 3.3, 3.11

[62] D. Hellmann. (2017, January) BaseHTTPServer — base classes for implementing web servers. [Online]. Available: http://pymotw.com/2/BaseHTTPServer 3.3.1.2

[63] PaulBoddie. (2012, April) BaseHTTPServer. [Online]. Available: http://wiki.python.org/moin/BaseBttpServer 3.3.1.2

[64] Rash. (2014, October) Python 3.x BaseHTTPServer or http.server. [Online]. Available: http://stackoverflow.com/questions/232654569/python-3-x-basehttpserver-or-http-server 3.3.1.2

[65] R. Hessmer. (2009, October) Kinematics for Lynxmotion robot arm. [Online]. Available: http://www.hessmer.org/uploads/RobotArm/Inverse%2520Kinematics%2520for%2520Robot%2520Arm.pdf 3.3.2.1, 3.3.2.1

[66] S. B. Niku, *Introduction to Robotics*. Pearson Education, Inc. 3.3.2.1

[67] T. Foote and M. Purvis. (2014, December) REP 103 – standard units of measure and coordinate conventions. [Online]. Available: http://www.ros.org/reps/rep-0103.html 3.3.2.1

[68] V. A. Deshpande and G. P. M., "Analytical solution for inverse kinematics of SCORBOT-ER Vplus robot," vol. 2, pp. 478–481, March. [Online]. Available: http://ijetae.com/files/Volume2Issue3/IJETAE_0312_83.pdf 3.3.2.1

[69] Gazebo. (2014) Gazebo tutorials. [Online]. Available: http://gazebosim.org/tutorials?cat=install 3.4

[70] baijuchaudhari. (2015, November) sboter4u. [Online]. Available: https://github.com/baijuchaudhari/sboter4u 3.4

[71] C. Christensen. (2017, April) Southern Arm Control Description. [Online]. Available: https://github.com/greenpro/sac_description 3.2, 3.6

[72] jwatson_utah_edu and P. Mitrano. (2016, August) How to programmatically attach link to a model during simulation? [Online]. Available: answers.gazebosim.org/question/14072/how-to-programmatically-attach-link-to-a-model-during-simulation/ 5.1

[73] M. Purvis. (2015, October) Unittesting. [Online]. Available: http://wiki.ros.org/UnitTesting 5.1