ECE - 478/578
“Intelligent Robots I”

Assignment 1: “Turtlebot2i for Paradise Lost Play”

Created by:
Sandeep Vankata
Dustin Schnelle

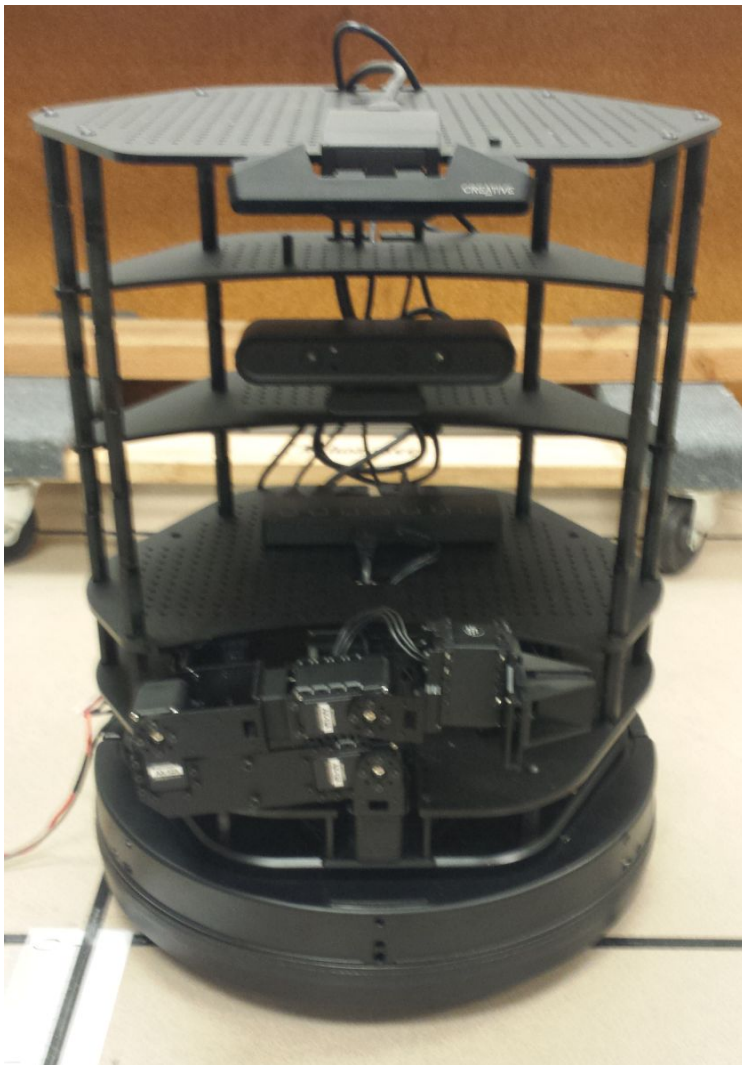


Table of Content:

Objective:

The goal of this report is to introduce both college and high school students to the Turtlebot2i and provide the steps necessary to perform basic task with the robot. Such as animations for the robot play “Paradise Lost.” This is a basic introduction to the Turtlebot2i and the interface for the animations created, so they can be reproduced and built upon by future classes. It will explain all the basics that you need to start using Turtlebot2i and it will also explain how to troubleshoot different issues that may occur during operation.

To accomplish the goal of introducing the students to the basic operation of the Turtlebot2i we will start with an introduction to the both the robot’s hardware and software. This introduction is not meant to give you full descriptions of Turtlebot software and hardware, but instead it’s meant to give you an overview. Next we will explain how to setup the Turtlebot2i and everything that is needed for getting started. After teaching you how to setup Turtlebot2i we will show you how to test the functionality of the robots movements/animations. This will also explain that techniques that were used by our team to create the movements/animation. Next we will explain the development of the functions used to control the movements/animation created for this project. Finally we will talk about issues that we faced and how to debug the issues when they occur.

I would like everyone reading this report to know that most of this content in the document is not original. We wanted to provide a report with all of the basic information and oversight needed to operate without getting to caught up in the technical details right away. Therefore this report is a summary of our research, documents that we used during the project, and experience working with Turtlebot2i. This report combine some of the great information provided by previous students experience with our own. The sources for this information can be found in the last page of the report. Please also note that this is the first report about using the Turtlebot2i for robot theater and there is still more that can be improved upon.

Introduction:

Turtlebot2i:

Turtlebot is a low-cost open-source robot development kit for applications on wheels. According to Turtlebot's website, "you'll be able to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications. The TurtleBot 2i extends upon previous iterations of the TurtleBot with a modular chassis and for the first time, native support of robotic arms. The TurtleBot 2i offers the Pincher MK3 4 DOF Robotic Arm as a fully supported standard option allowing the robot to interact with small objects in the real world, effectively transforming the TurtleBot into an extremely capable mobile manipulator. Overall the Turtlebot2i is one of the most popular educational robotic platforms available.

The Turtlbot2i's in the robotics lab consist of a Kobuki mobile base, 2D/3D distance sensors, Intel Joule 570x chip, Turtlebot mounting hardware kit, and Pincher MK3 4 DOF Robotic Arm. As is shown in the picture #1 below.

Picture #1: Turtlebot2i turned off with robotic arm extended

Hardware Modules:

Listed below are the different hardware modules that make up Turtlebot 2i's design. The main modules consist of the CPU, sensors, robotic arm, and mobile base. The description of the hardware used by the mobile arm is in the following section.

CPU:

- INTEL NUC - BOXNUC6CAYH
- 8GB Ram
- 120GB or better SSD
- 802.11AC WiFi / Bluetooth 4.0
- Ubuntu 16.04 / ROS Kinetic

Sensors:

- Intel RealSense 3D Camera SR300-Series
- Orbbec Astra Cam
- Accelerometer/Gyro/Compass
- Edge Detection & Bumper Sensors

Mobile Robot:

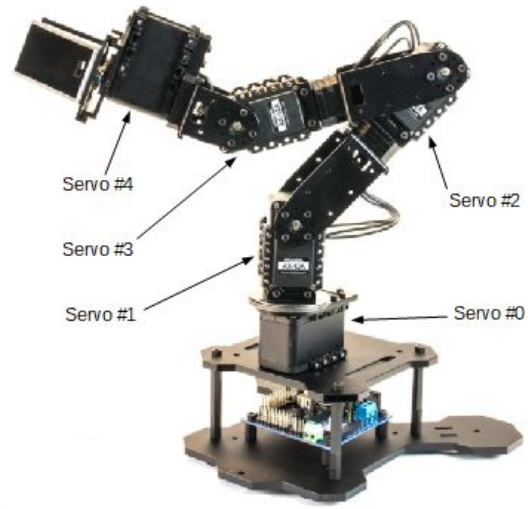
- Kobuki Mobile Base
- Modular & Interchangeable Decks
- Pincher MK3 Robo Arm
- Arbotix-M Robocontroller
- Maximum translational velocity: 70 cm/s 13
- Maximum rotational velocity: 180 deg/s (>110 deg/s gyro performance will degrade)
- Payload: 2kg (without arm), 1kg (with arm)
- Cliff: will not drive off a cliff with a depth greater than 5cm
- Threshold Climbing: climbs thresholds of 12 mm or lower
- Rug Climbing: climbs rugs of 12 mm or lower
- Expected Operating Time: 4-6 hours (operating time varies depending on loadout)
- Expected Charging Time: 2-3 hours (charge time varies depending on loadout)
- Docking/Charging Station: automatic within a 2mx5m area in front of the docking station

The Turtlebot 2i's hardware mainly consists of embedded modules that are based on ROS (Robotics Operating Software) robotics software platform. It's equipped with the Intel Joule 570x chip, accelerometer/gyro/compass, edge detection, bumper sensors, and RealSense 3D camera. All of these modules and others enable the robot to perform many tasks, such as autonomous driving, obstacle detection, and obstacle avoidance in real time.

Pincher MK3 4 DOF Robotic Arm:

The addition of the Pincher MK3 4 DOF robotic arm adds extra functionality to the Turtlebot2i. The arm can be controlled by an "open source software" called Moveit and allows

the robot to acquire new capabilities such as object sorting, and object manipulations. The PhantomX Pincher MK3 Robot Arm is a 4 degree-of-freedom robotic arm and an easy addition to the TurtleBot ROS robot platform. This hardware kit comes with everything needed to physically assemble and mount the arm as a standalone unit or as an addition to your Turtlebot Robot/mobile platform. Vanadium Labs has developed an ROS stack that makes communication with the robotic arm a breeze. The PhantomX Pincher MK3 Robot Arm is shown by itself with the labeled servos in picture #2 below. The reason there is only 4 degrees of freedom and 5 servos is because the last servo (servo #4) is used for the gripper on the robot arm.



Picture #2: Pitcher MK3 4 DOF robotic arm with servos labeled

Software Modules:

The Turtlebot2i is reliant on two type of software in order to produce motion and animations. It uses both ROS (Robotic Operating System) and Moveit software. To learn more about ROS is recommend reading the first 5 pages of “Gentle Introduction to ROS” and check out the ROS wiki. Both of which are listed in the reference section of this report.

ROS (Robotic Operating System) :

As is stated on the ROS wiki, “ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.” The Turtlebot2i uses Arbotix_ROS version 0.11.0.

Moveit:

MoveIt! comes with a plugin for the ROS Visualizer (RViz). The plugin allows you to setup scenes in which the robot will work, generate plans, visualize the output and interact directly with a visualized robot. We will explore the plugin in this tutorial.

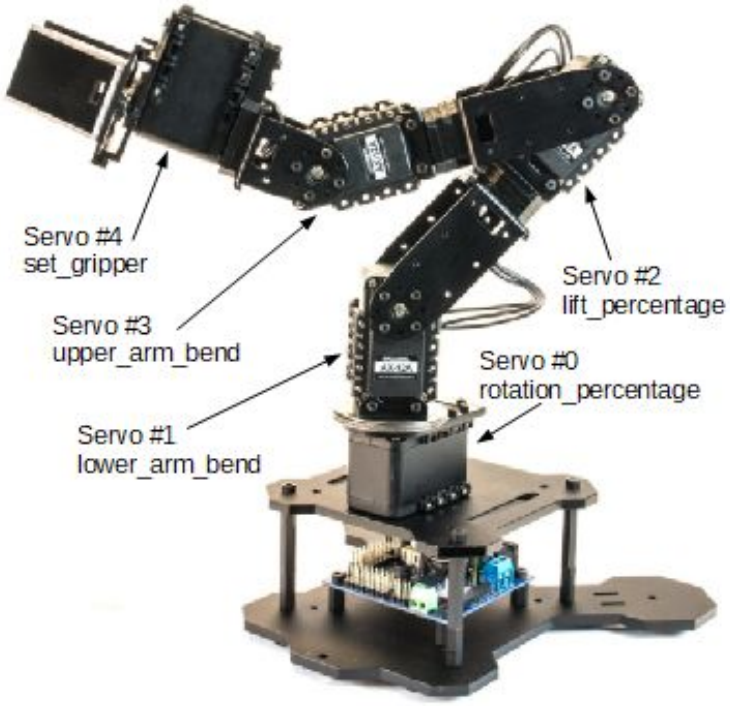
In MoveIt!, the primary user interface is through the RobotCommander class. It provides functionality for most operations that a user may want to carry out, specifically setting joint or pose goals, creating motion plans, moving the robot, adding objects into the environment and attaching/detaching objects from the robot.

Interfaces Using Moveit to Control the Arm

We had developed the following interfaces using moveit to control the robotic arm. Since the robotic arm has 4 “degrees of freedom” we had to develop at least 4 different functions to control the 4 servos in a safe way. However there are a total of 5 servos when you include the gipper servo, so a total of 5 functions needed to be created. Listed in table #1 below is a list of the 5 initial functions and what servo they control. They’re also shown in picture #3 to provide better visualization of the functions and there servos.

Function Name	**Servo Number**
rotation_percentage()	Servo #0
lower_arm_bend()	Servo #1
lift_percentage()	Servo #2
upper_arm_bend()	Servo #3
set_gripper(state)	Servo #4

Table #1: Function names and the associated servo



Picture #3: Pitcher MK3 4 DOF robotic arm with servos labeled and functions shown

Those 5 initial functions are not enough to safely control the robotic arm without doing damage to the servos. The 5 functions listed in table #1 only control single servos, so we need to create a function that would provide both a safe known starting position and ending position. Those two safe positions were then coded into the reset_arm() function and intial_pos() function. The reset_arm() function is used at the end of every animation to insure that the arm is in a safe known location when the next animation is executed. Whereas the intial_pos() function is used at the beginning of every animation to insure that the arm is in a known location at the start of every animation. Using a sequential combination of these functions is what was used to create animation for the Turtlebot2i. Listed below are pictures of the various functions and how they were designed in greater detail.

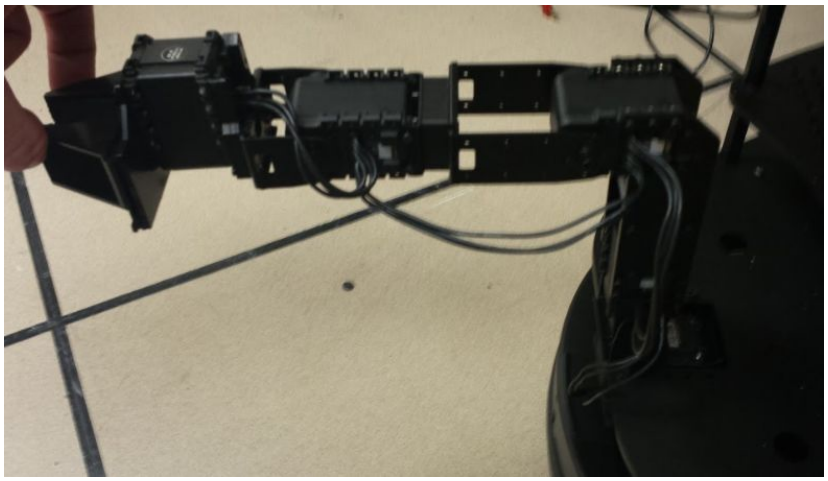
reset_arm() function

The reset_arm() function is used to place the robotic arm back to a safe known starting place after the end of an animation. This function takes no parameters and is able to modify the "turtlebot" object using *"self"* in python. This is the position command that should end any animation created. A picture of the reset position is shown below in picture #4. Make sure this is the position the arm is in when beginning operation.

Picture #4:

intial_arm() function

The intial_arm() function is used to place the robotic arm into a safe known starting place at the beginning of an animation. This function takes no parameters and is able to modify the “turtlebot” object using *“self”* in python. This is the position command that should begin any animation created. This function gives all of the servos a position value near 0. If the function fails then it will print “Go failed for right up.” We will talk about ways to fix this error in the debugging mechanism section of the report. Once in the initial “right_up” position other functions can be used in combination to create animations.



Picture #5

display() function

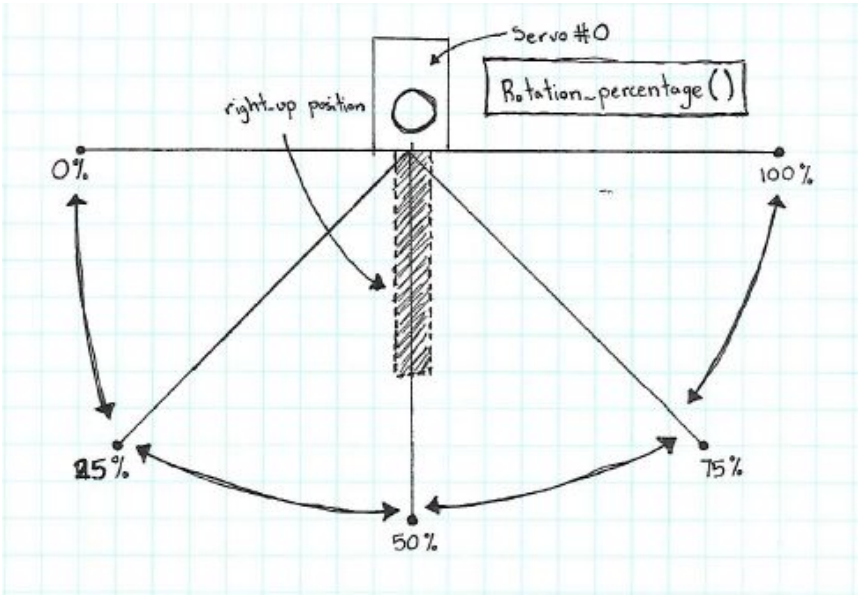
The display() function is mainly use for testing/debugging during the creation of animations. The function shows the planning frame and current state of the servos with associated group names, which can be used when trying to create new motion functions for the Turtlebot2i. This function takes no parameters and is able to read the “turtlebot” object’s current status using *“self”* in python.

set_gipper() function

The set_gipper() function is used to control servo #4 and the opening/closing of the gripper on the robotic arm. This function can be used to gip and move different items from the surrounding environment. This function requires one parameter called “state” to control the current state of the gripper. It’s able to modify the “turtlebot” object by using *“self”* in python. There are four states for the gripper opened, close, neutral, and tighten. The opened state is used to fully open the gripper and the close state is used to fully close the gripper. The neutral state is used to put the gripper in a position between opened and close. The final state tighten is used to close the gripper in incremented steps, which is great when programming the gripper to grab objects.

rotation_percentage() function

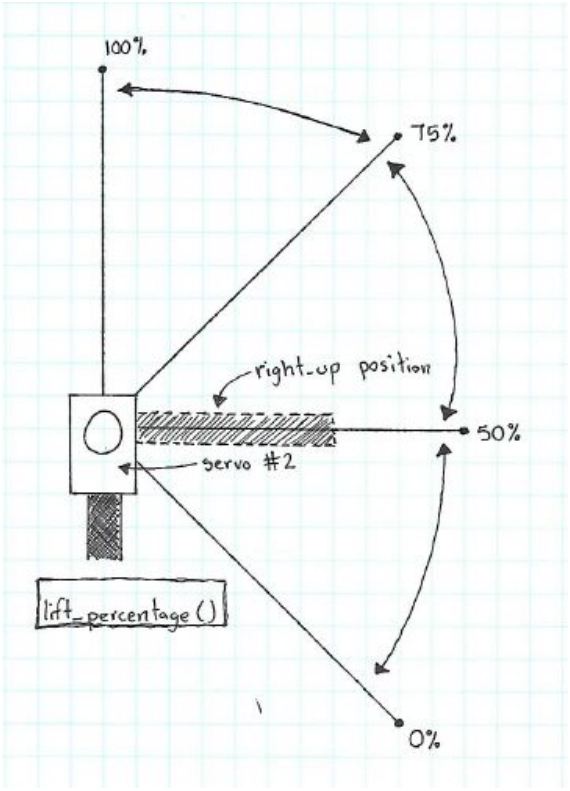
The rotation_percentage() function is used to control servo #0 and the left/right rotation of the robotic arm. This is the only left/right motion that the Pincher MK3 4 DOF robotic arm is capable of, so it’s used quite often to rotate the arm into the correct positioning.This function requires one parameter called “percentage” to control the current state of servo #0. It’s able to modify the “turtlebot” object by using *“self”* in python. The function was designed to rotate the Turtlebot2i’s robotic arm from left to right using a percentage between 0-100%. When the percentage is equal to 0% the arm is rotated to the far left boundary and when the percentage is equal to 100% the arm is rotated to the far right boundary. Picture #6 below shows the behavior of the function between a range of 0-100%. The shaded right_up position is where the servo’s position will locate itself after the intial_arm() function.



Picture #6: Functionality of the rotation_percentage function

lift_percentage() function

The lift_percentage() function is used to control servo #2 and part of the up/down motion of the robotic arm. This is one of three up/down motions that the Pincher MK3 4 DOF robotic arm is capable of, so it needs to be used carefully when using both the upper_arm_bend and lower_arm_bend functions. This function requires one parameter called “percentage” to control the current state of servo #2. It’s able to modify the “turtlebot” object by using *“self”* in python. The function was designed to lift the Turtlebot2i’s robotic arm up and down using a percentage between 0-100%. When the percentage is equal to 0% the arm is moved down to the far lower boundary (touching the floor with the gripper) and when the percentage is equal to 100% the arm is moved down to the far upper boundary. Picture #7 below shows the behavior of the function between a range of 0-100%. The shaded right_up position is where the servo’s position will locate itself after the intial_arm() function.

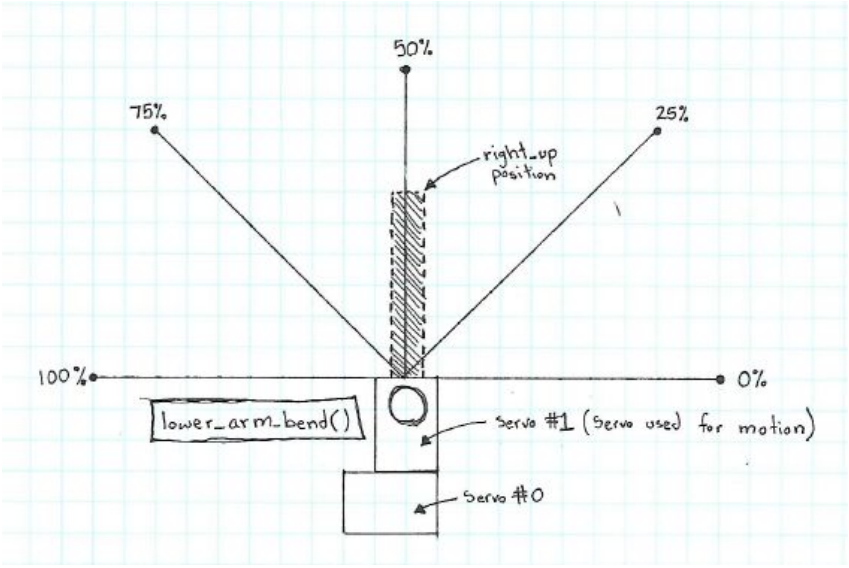


Picture #7: Functionality of the lift_percentage function

lower_arm_bend() function

The lower_arm_bend() function is used to control servo #1 and part of the up/down motion of the robotic arm. This is one of three up/down motions that the Pincher MK3 4 DOF robotic arm is capable of, so it needs to be used carefully when using both the upper_arm_bend and lift_percentage functions. This function requires one parameter called “percentage” to control the current state of servo #1. It’s able to modify the “turtlebot” object by using *“self”* in

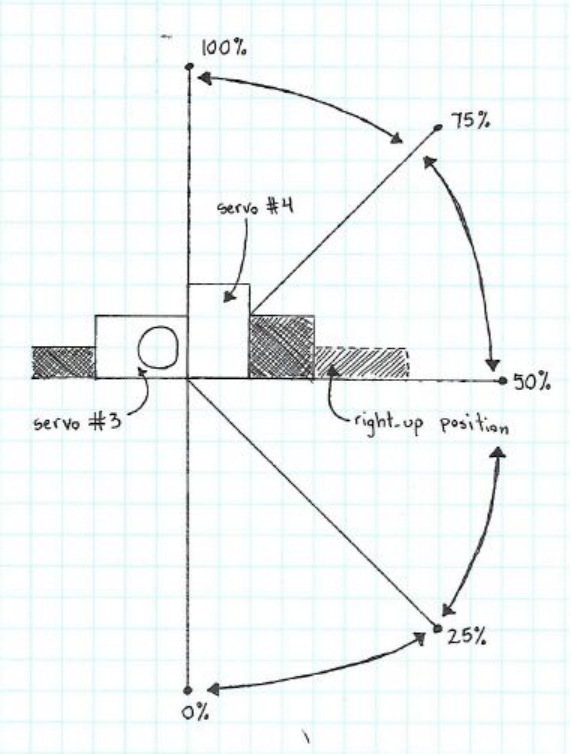
python. The function was designed to lift the Turtlebot2i's robotic arm up and down at the lower joint of the arm using a percentage between 0-100%. When the percentage is equal to 0% the arm is moved down to the far lower boundary (touching the floor with the gripper) and when the percentage is equal to 100% the arm is moved down to the far upper boundary. Picture #8 below shows the behavior of the function between a range of 0-100%. The shaded right_up position is where the servo's position will locate itself after the intial_arm() function.



Picture #8: Functionality of the lower_arm_bend function

upper_arm_bend() function

The upper_arm_bend() function is used to control servo #3 and part of the up/down motion of the robotic arm. This is one of three up/down motions that the Pincher MK3 4 DOF robotic arm is capable of, so it needs to be used carefully when using both the lower_arm_bend and lift_percentage functions. This function requires one parameter called "percentage" to control the current state of servo #3. It's able to modify the "turtlebot" object by using *"self"* in python. The function was designed to lift the Turtlebot2i's robotic arm up and down at the upper joint of the arm using a percentage between 0-100%. When the percentage is equal to 0% the arm is moved down to the far lower boundary and when the percentage is equal to 100% the arm is moved down to the far upper boundary. Picture #9 below shows the behavior of the function between a range of 0-100%. The shaded right_up position is where the servo's position will locate itself after the intial_arm() function.



Picture #9: Functionality of the upper_arm_bend function

Getting Started:

Assumptions:

- Kuboki base is fully charged.
- The LiPo battery is fully charged or switch power supply is connected
- Robot is connected to the WiFi network (PSU network)

Note: The Turtlebot2i has used the ROS network in the past, but we only used the PSU network during the project.

This section provides instructions on how to use the Turtlebot2i for the first time. Do not worry if you don't completely understand every detail and why we're issuing the commands in two computers. This will be talked about in the last part of this section "Code Setup and Build Instructions." The more that you use Turtlebot2i the more you will understand why these instructions and commands are necessary.

Hardware Setup:

Power on sequence:

You need to follow the following instructions in order to power up the Turtlebot2i. In order to obtain the best overall functionality during creating motions and testing it's best to use an switching power supply to give direct power to the Turtlebot2i. The fully charged LiPo battery doesn't always have enough power to produce the proper motions all the time. If you are using the Turtlebot at the workstation then the HDMI and USB connectors need to be plugged into the Intel Joule before the power on sequence is initiated.

1. Disconnect the chargers and remove cables (optional if battery not charged).
2. Turn on the base switch located on the Kubuki base and wait 10 seconds.
3. Turn on the Joule by switching the metal switch and wait about 45 seconds.
   Note: Time is needed to allow the OS to bootup.

Charging the Turtlebot2i:

The Turtlebot2i is equipped with two LiPo batteries needed for operation. The Kobuki base has a LiPo battery located in a compartment in the bottom of the base. The Kubuki base battery is charged using the charging stations shown in picture #10 below.

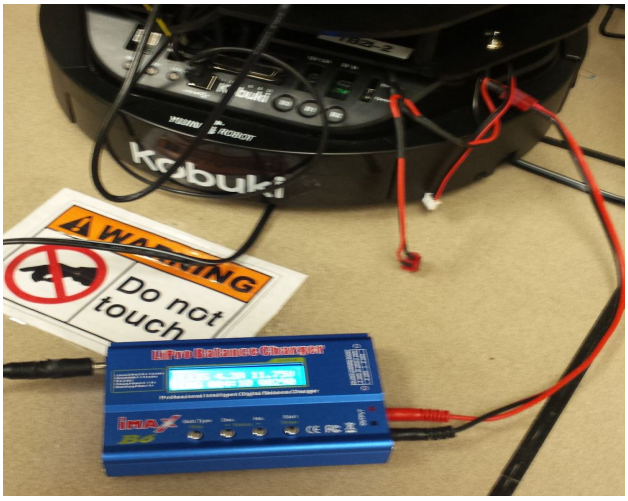
Picture #10:

The charging station has a port for a switching power supply to be connected to as is shown in picture #11 below.

Picture #11:

Once the charging station is connected correctly a orange/red LED will come on on the charging station. The Turtlebot2i needs to have its robotic arm facing the LED on the charging station. The light turns off once you place the Turtlebot2i on the charging port. You can tell that the base is charged if the status LED on the Kobuki base lights up green. It requires about 30 minutes to fully charge the Turtlebot2i's base and can be safely charged while being unattended.

The second LiPo battery can be seen on the Turtlebot2i by looking under the Intel Joule 570x integrated development platform. The battery is a 11.1V LiPo battery that has the option of being plugged into a switching power supply (the same one being used by the HR0S1 "Jimmy") or it can be charged up using the iMax B6 LiPro Balance Charger as is shown in picture #12 below. The LiPro Balance Charger need to be setup for LiPo battery charging using 11.1V and 4A. Once the configuration of the charger is complete press and hold the start button. Then press enter one more time on the charger to begin charging. You must be present during the charging of this LiPo battery because it's not safe to leave the battery charging unattended. A full charge will take about 30 - 45 minutes, so plan accordingly.



Picture #12:

The LiPo battery located under the Intel Joule board is all that is need to obtain operation from just the Pitcher MK3 4 DOF robotic arm. This can be done by unplugging the USB hub and connecting the USB for the servo controller directly to the Intel Joule USB port. Otherwise both batteries need to be charger in order to get full operation from the Turtlebot2i.

Software Setup:

Code Setup:

Code setup and build instructions for ROS with turtlebot2i

```
source /opt/ros/kinetic/setup.bash

cd ~

mkdir -p ~/turtlebot2i/src

cd ~/turtlebot2i/src

git clone https://github.com/Interbotix/turtlebot2i.git .

git clone https://github.com/Interbotix/arbotix_ros.git -b turtlebot2i

git clone https://github.com/Interbotix/phantomx_pincher_arm.git

git clone https://github.com/Interbotix/ros_astra_camera -b filterlibrary

git clone https://github.com/Interbotix/ros_astra_launch

cd ~/turtlebot2i
```

```
catkin_make
```

Bashrc Setup:

- **You will need to place the following at the bottom of your ~/.bashrc file**
- **Edit .bashrc with your choice of editor (gedit, nano, vi)**
- **Be sure to replace example.hostname with your computer's hostname**

```
source /opt/ros/kinetic/setup.bash

alias goros='source devel/setup.sh'

export ROS_HOSTNAME=example.hostname

export TURTLEBOT_3D_SENSOR=astra

export TURTLEBOT_3D_SENSOR2=sr300

export TURTLEBOT_BATTERY=None

export TURTLEBOT_STACKS=interbotix

export TURTLEBOT_ARM=pincher
```

Setup the python moveit interface:

```
mkdir -p ~/turtlebot2i/work cd $_

git clone https://github.com/cvsandeep/Turtlebot.git .
```

Bring Up:

This are the basic things to run after every reboot

cd ~/turtlebot2i

goros

roscore &

To bring up the full robot with mapping, arm pose manager, moveit and zone detection:

roslaunch turtlebot2i_bringup turtlebot2i_demo1.launch

We use the following command to only bring up the ARM for using moveit

roslaunch phantomx_pincher_arm_bringup moveit.launch

Testing:

Testing the bot with the following demos

- roslaunch turtlebot_teleop keyboard_teleop.launch
- roslaunch turtlebot2i_block_manipulation block_sorting_demo.launch

Testing the arm with moveit using python:

We had created a test script to test all the movements of the ARM

```
$roslaunch phantnomx TBD

$cd ~/turtlebot2i/work/Turtlebot

$Python test.py
```

Planning using Joint control mechanism:

APi's developed to control joints using moveit python:

This section of the of the report will show the application programming interface that we developed to control the servos of the robotic arm. The API was created using Moveit Python which allowed to the coding of the function of motion and animation to be coded in Python. The reason for doing so was because it was easy to test the python code as it was being developed by running the experimental python code from the bash window of the Turtlebot2i. The animations we created are dependent on the motions that were created and shown in the "Developed the Interface Using moveit To control the arm" section of the report. This section will explain the basic animations created for Turtlebot2i to perform in the play "Paradise Lost." It will also talk about how reverse kinematics us used to duplicate different animations from existing animations and motions. Lastly it will talk about the motion control of the Turtlebot2i and the steps to create your own animations.

Animations:

The animations described below were created using a sequential combination of motion functions create. These animations were specifically created for the robot theater play "Paradise

Lost," but they could be repurposed for other future plays and projects. The are animations for building things, interacting with the other robots, and showing emotion.

Basic motions developed:

For the production of the robot theatre play "Paradise Lost" we created a total of 9 different animation. The animations were created through a combinatorial sequence of the motions created for each servo in a previous section of the report and need to be used as a reference in order to understand the motions of the different animations. We started off by building the smaller/simpler animations first and then were able to build more complex animations from the simpler ones. Some reverse kinematics were used in order to create some of the more complex animations as well and will be talked about later in this section. The goal in creating the animations in this way is that we could easily teach other users of the Turtlebot2i to create more complex animations from our motion library. An important thing to notice is that every animation starts with the function intial_pos() and ends with the function reset_pos(). The various animations below will give a brief summary of the animation as well as how it was created.

Introduction:

The introduction animation was created using the lift percentage function to produce a wavelike motion of the robotic arm. This animation is used to show that that robotic arm works, it can also move, and assemble another robot on the stage. It also uses the screw motion and build robot animations. The introduction animation is a perfect example of reusing other animations to create more complex motions. In this case we needed to show the Turtlebot2i robot arm building something on the stage, so it makes sense to reuse both the screw motion and build robot functions. The python code for the introduction animation are shown below in picture #13.

```
def introduction():
    print "----------------Introduction----------------"
    arm.intial_pos()
    arm.lift_percentage(75)
    arm.lift_percentage(25)
    arm.lift_percentage(50)
    arm.lift_percentage(75)
    arm.lift_percentage(50)
    arm.reset_arm()
    screw_motion()
    build_robot()
```

Picture #13: Introduction python animation code

Wave:

The wave animation is an extremely simple animation that places the arm into the waving position and then uses a for loop to rotate the robotic arm right and left. This kind of

simple animation can be used to build larger more complex animations. This was the animation that we decided to use for the confusion of God played by Turtlebot during the last scene of “Paradise Lost.” Picture #15 below shows the python code for the wave animation.

```
def wave():
    print "----------------wave----------------"
    arm.intial_pos()
    arm.lower_arm_bend(20)
    arm.upper_arm_bend(90)
    for num in range(1,5):
        arm.rotation_percentage(40)
        arm.rotation_percentage(60)
    arm.reset_arm()
```

Picture #15: Wave python animation code

Board Replace:

The board replace animation is used for the creation of an new robot (Eve). It’s different from the build animals animation because it uses the gripper to move a microcontroller from one robot to another robot. This is another one of the animations used to create the more complex introduction animation. The biggest thing to take away from all of the animations is the fact that they all start with the intial_arm() function and end with the reset_arm() function. This is important when creating your own animations. Picture #16 below shows the python code for the board replace animation.

```
def board_replace():
    print "----------------board_replace----------------"
    arm.intial_pos()
    #Picking the board
    arm.lift_percentage(75)
    arm.set_gripper("opened")
    arm.rotation_percentage(10)
    arm.upper_arm_bend(0)
    arm.lift_percentage(50)
    arm.lift_percentage(45)
    arm.set_gripper("close")
    arm.lift_percentage(85)
    #Placing the board
    arm.rotation_percentage(60)
    arm.lift_percentage(50)
    arm.set_gripper("opened")
    arm.intial_pos()
    arm.reset_arm()
```

Picture #16: Board replace python animation code

Build Animals:

The build animals animation is very similar to the build robots animation except the height at which the robot arm builds is a higher value. This is because the animals are taller than the HR0S1’s in the laying down position. Notice that this animation also has a parameter called rotation and it’s used to change the rotation position of the animation. The reason we designed the animation in this way was because we didn’t know exactly where the animals would be located on the stage. It was important that we could change the rotation quickly.

Adding parameters to animations like build animals provides even more freedom when it comes to performing the animation. This might be the better approach when designing new animations for future theatre productions. Picture #17 shows the python code for the build animals animation.

```
def build_animals(rotation):
    print "----------------build_animals----------------"
    arm.intial_pos()
    arm.rotation_percentage(rotation)
    arm.set_gripper("opened")
    arm.set_gripper("close")

    arm.upper_arm_bend(40)
    arm.lift_percentage(40)
    arm.set_gripper("opened")
    arm.set_gripper("close")
    arm.lift_percentage(70)
    arm.set_gripper("opened")
    arm.set_gripper("close")
    arm.intial_pos()
```

Picture #17: Build animals python animation code

Guide:

The guide animation is used to guide a HR0S1 robot during the play. This is one of our animations that would probably benefit from a parameter to control the speed of the rotational motion. The animation slowly rotates the robotic arm for left to right using both a for loop and the rotation_percentage function. This is also one of our simpler animations that can be built upon to create more controlled complex animations. Picture #18 shows the python code for the guide animation.

```
def guide():
    print "----------------guide----------------"
    arm.intial_pos()
    arm.set_gripper("opened")
    arm.lower_arm_bend(20)
    arm.upper_arm_bend(0)
    arm.rotation_percentage(0)
    arm.set_gripper("close")
    for num in range(0,20):
        arm.rotation_percentage(num*3)
    arm.set_gripper("opened")
    arm.reset_arm()
```

Picture #18: Guide python animation code

Press Enter:

The press enter animation is one of our more complex animations. It reuses some of the motions used to create the screw motion with some minor changes. The animation rotate the Turtlebot's arm to location where a enter key is located. It then squeezes and releases the the enter key in order to simulate pressing the button. Like I stated earlier a lot to the motions used by screw motion were used and modified in order to create the press enter animation. Picture #19 shows the code for the press enter animation.

```
def press_enter():
    print "----------------press_enter----------------"
    arm.intial_pos()

    #pick the screw

    arm.upper_arm_bend(0)

    # this may change according to placement of key board
    arm.rotation_percentage(90)

    #According to placment TWEAKKKKKKKKKK
    arm.lift_percentage(70)
    arm.lower_arm_bend(70)
    arm.set_gripper("close")
    time.sleep(2)
    arm.set_gripper("opened")
    arm.lower_arm_bend(50)
    arm.lift_percentage(50)
    arm.intial_pos()
    arm.reset_arm()
```

Picture #19: Press enter python animation code

Bless:

The bless animation is another simple animation that uses three different for loops to perform a blessing motion in three different rotational positions. This animation was used to bless the animals during the parade of animals scene in “Paradise Lost.” Picture #22 shows the python code for the bless animation.

```
def bless():
    print "---------------bless----------------"
    arm.intial_pos()
    for num in range(1,3):
        arm.lower_arm_bend(20)
        arm.upper_arm_bend(80)
        arm.upper_arm_bend(30)
    arm.rotation_percentage(100)
    for num in range(1,3):
        arm.lower_arm_bend(20)
        arm.upper_arm_bend(80)
        arm.upper_arm_bend(30)
    arm.rotation_percentage(0)
    for num in range(1,3):
        arm.lower_arm_bend(20)
        arm.upper_arm_bend(80)
        arm.upper_arm_bend(30)
    arm.reset_arm()
```

Picture #22: Bless python animation code

Inverse Kinematics Used to Develop the Animations:

Build Robot:

The build robot animation was created using 4 unique motion functions to make it look like the Turtlebot2i’s arm is assembling something. The robotic arm moves back and forwarth while opening and closing the gripper in order to create this animation. As was shown in the

introduction python code in picture #13 this animation was used to create more complex animations. The python code for build robot is shown in picture #14 below.

```
def build_robot():
    print "----------------build_robot----------------"
    arm.intial_pos()
    arm.upper_arm_bend(0)
    arm.lift_percentage(35)
    arm.set_gripper("opened")
    arm.set_gripper("close")
    arm.lift_percentage(50)
    arm.rotation_percentage(40)
    arm.lift_percentage(35)
    arm.set_gripper("opened")
    arm.set_gripper("close")
    arm.lift_percentage(50)
    arm.rotation_percentage(5)
    arm.lift_percentage(35)
    arm.set_gripper("opened")
    arm.set_gripper("close")
    arm.lift_percentage(50)
    arm.intial_pos()
    arm.reset_arm()
```

Picture #14: Build robot python animation code

Screw Motion:

The screw motion animation has to be the most complex animation created for this project. The reason it was so complex is because is animation would pick up a screw from a known location. Then it will move the screw to the building area and create animation that make it appear that the robot arm is assembling something with the screw. Lastly the arm will carefully place the screw back to the known location, so it can be used again. This is definitely the longest of the animations created and the reason for that is because the screw had to be carefully picked and placed. This required a slowly incrementing lifting sequence of the lift_percentage and lower_arm_bend functions which can be seen in the python code shown in picture #20 and picture #21 below. This animation is also an example of when inverse kinematics was used to create an animation. First we focus on just getting the robotic arm to pick-up the screw and move it to the desired location. Then performing those motion functions in reverse order allow for the animation to accurately place the screw back without as much testing of the animation.

```
def screw_motion():
    print "----------------screw_mootion----------------"
    arm.intial_pos()

    #pick the screw

    arm.set_gripper("opened")
    arm.upper_arm_bend(0)

    # this may change according to placement of key board
    arm.rotation_percentage(100)

    #According to placment TWEAKKKKKKKKKK
    arm.upper_arm_bend(3)
    arm.lift_percentage(70)
    arm.lower_arm_bend(69)
    arm.set_gripper("close")
    for num in range(0,13):
        arm.lift_percentage(71 - (num*2))
        arm.lower_arm_bend(70 - (num*2))



    #screw motion
    arm.intial_pos()
    arm.lift_percentage(45)
    arm.upper_arm_bend(0)
    time.sleep(1)
    for num in range(1,5):
        arm.rotation_percentage(40+(num*3))
    for num in range(1,5):
        arm.rotation_percentage(55-(num*3))
```

Picture #20: Part 1 of screw motion python animation code

```
    # place the screw back
    arm.lift_percentage(50)
    arm.lower_arm_bend(45)
    time.sleep(1)

    #Copied from above replace values
    arm.upper_arm_bend(0)

    # this may change according to placement of key board
    arm.rotation_percentage(100)

    #According to placment TWEAKKKKKKKKKK
    arm.upper_arm_bend(3)
    for num in range(1,8):
        arm.lift_percentage(55 + (num*2))
        arm.lower_arm_bend(57 + (num*2))

    arm.set_gripper("opened")
    arm.reset_arm()
```

Picture #21: Part 2 of screw motion python animation code

Steps to Create Your Own Animations:

cd ~/turtlebot2i/work/Turtlebot

$python

Import a tb2i module and create an instance of it

```
import tb2i as TB
arm = TB.tb_arm()
```

Using the arm instance you can use the api's to control the arm.
For eg to go to initial pos use
    arm.intial_pos()
For eg to go to reset pos use
    arm.reset_arm()
For eg to close the gripper use
    arm.set_gripper("close")

Bot Movement Control:

To control the bot run this commands and follow the on screen instructions to control the bot to the desired direction.

cd ~/turtlebot2i

goros

roscore &

roslaunch turtlebot_teleop keyboard_teleop.launch

Debugging mechanism:

While working with Turtlebot2i to create animations for “Paradise Lost” we ran into many problems and this section will talk about some of those problems. This will also talk about ways that we found to debug the issues. Some of the issues discovered were due to hardware and others were because of software. When it comes to the hardware the number one issue that we encountered was servos misbehaving. This could come in the form of the robotic arm not responding to command give from the functions or one servo locking up during motion. There were also a couple of times when all of the servos would freeze while in the middle of animation. This was something that we had to work around while developing and staging the animations.

There were couple of different fixes that we came up with for the problematic servos. We could unplug the servo communication wiring and let the arm essentially turn off. Then give the Turtlebot the intial_arm command, so it would reinitialize itself to the right up (safe position). Although this solution didn’t always work and in those cases we would have to completely power cycle Turtlebot2i. Power cycling the robot is very time consuming, so we developed a shortcut command called “start” to start both goros and roscore. We also created a shortcut command called “tasl” to direct you to the folder the python code is run from.

The first couple of weeks that we work with Turtlebot2i were hard because we didn’t have a full understand of how Turtlebot2i functioned. It took us some time to figure out how to properly charge the robot and how to use moveit. We hope that this report allows you to have a better understanding of Turtlebot2i, so you can create future improvements to the robot theatre.

Future updates needed:

This section will talk about the future updates that need to be made to both Turtlebot2i and the overall structure of a robot theatre. As was state in the section about the animation of the robotic arm improvements can be made to the animations by adding in parameters to the functions, so they’re more versatile animations. It was not always easy to coordinate with other groups all the time, so having more flexibility in the animations would of produced better interactions between the robots. More motion functions could be added to the library as well like functions that control two servos during a single motion function. It would also be nice if the whole class worked on the same play because the coordinating or groups wasn’t always easy.

References:

- Turtlebot2i wiki: https://github.com/Interbotix/turtlebot2i
- ROS wiki: https://wiki.ros.org/ROS/Introduction
- A Gentle Introduction to ROS, Jason M. O’Kane, Independently published, Oct 2013, ISBN=9781492143239, https://cse.sc.edu/~jokane/agitr/
- Turtlebot2i Company Website: http://www.turtlebot.com/turtlebot2/

- Trossen Robotics PhantomX Pincher Robot Arm Kit Mark II - Turtlebot Arm: http://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx
- Turtlebot2i Final Report, Forrest Kaminski, Mohamed Abidalrekab, Sam Salin, Spring 2017 (Report provided by ECE-478 “Intelligent Robots I”, by Dr. Perkowski)
- Introduction to Turtlebot2i, Luis Santiago (Report provided by ECE-478 “Intelligent Robots I”, by Dr. Perkowski)