Manipulating the HiWonder Armpi Pro

Robotic Manipulation Final Report

August Halbert
ECE 3641
University of Michigan-Dearborn
Dearborn, Michigan
zmhalb@umich.edu

Raymundo Rodriguez
ECE 3641
University of Michigan-Dearborn
Dearborn, Michigan
rraymund@umich.edu

***Abstract*—The HiWonder Armpi Pro offers a wide variety of functionality to challenge and push what has been taught over the course of ECE 3641. The Armpi Pro comes with modules and demos that instruct you on how to work with different aspects of the robot. These demos, along with the concepts of forward and inverse kinematics, virtual machines, Robot Operating Systems (ROS), and python, allowed us to create our own unique script for the robot to perform.**

I. Introduction

Throughout the course of this semester, we have used the HiWonder Armpi Pro in order to learn the basics of robotic manipulation. For this project, we wanted to push this knowledge further, and see what we could accomplish by incorporating what we have learned in class to the more advanced demos. We began with the idea of programming a dance for the robot to perform, using forward and inverse kinematics to help us understand the path the robots arm was taking. Using the app, we were able to control the movement of the joints in real time, essentially modeling forward kinematics by altering the joint angles to create motion. Diving into the demos, we were able to figure out how to use NoMachine to connect wirelessly to the robot, and use the command line terminal to run the script we create. The major issue we encountered at this point was determining how to create our own script, how to run it on the Raspberry Pi, and most importantly, what commands we would need in order to control the robot. By using the documents provided with the demos, we were able to find source code that we could then build upon to create our own script. This also allowed us to be able to run our script on the robot itself. This project has been very interesting and informative. We have gone through a lot of trial and error to get to this point, as neither of us were familiar with Raspberry Pi, python, or virtual machines before this course. All of the hardships we went through to get this program to work enhanced our skills as future engineers, and the core concepts behind this project (inverse kinematics, robotic manipulation, virtual machines, python) are necessary for many industrial and engineering fields.

II. Related Work/Literature Review

The mecanum wheels on the robot allow it to have omnidirectional movement, which has been revolutionary for wheeled robot applications. The structure of the wheels allow for 360° movement even in tight spaces, making it ideal for places like warehouses and factories that may require moving large loads in tight spaces.

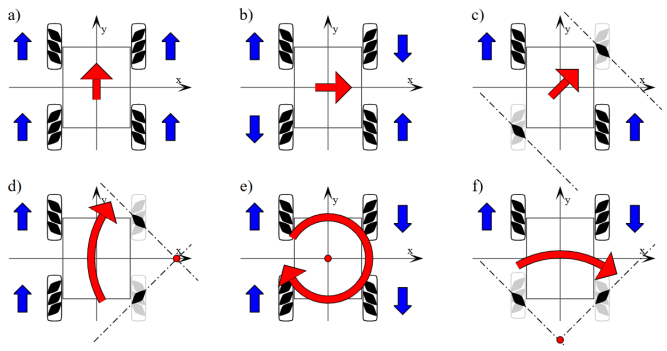


Fig. 1. Motion of Mecanum Wheels

Figure 1 shows how the mecanum wheels work together to create these movements. By orienting the rollers at 45°angles, the robot will move in the direction of the resulting vector summation of the wheels. Combining the increased movement capabilities the mecanum wheels provide and the gripper arm creates a robot that has a wide range of motion and interaction with its environment.

III. Method

A. Setup

In order to use the Hiwonder ArmPi Pro robot within this project, you need to have the **No-Machine** software in order to connect to another machine/computer. This allows the user to remotely connect to the robots Raspberry Pi operating system. This is done through wireless connection by connecting to the wifi hotspot put out by the robot. Once connected to the wifi, the ubuntu remote desktop will appear on NoMachine.

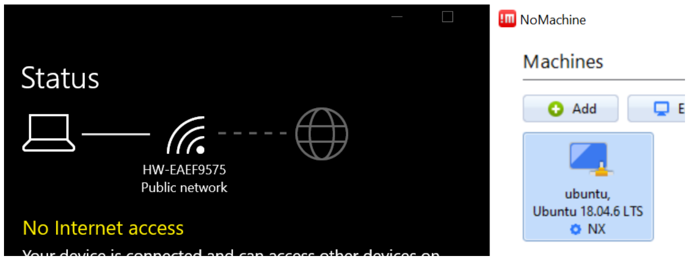


Fig. 2. Connecting to Armpi-Pro

Figure 2 shows the wirelesses connection to the robot and it showing up as a remote desktop in No-Machine.

Now that we are wirelessly connected to the Raspberry Pi, we have access to the Raspberry Pi's operating system, as it is its own computer.

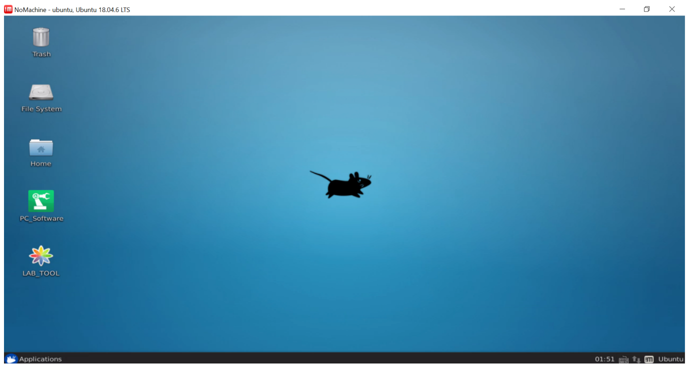
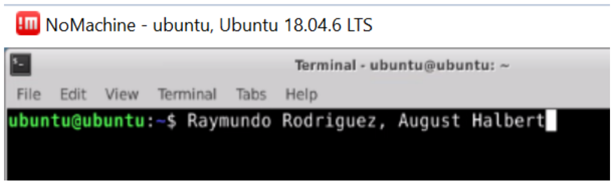


Fig. 3. Raspberry Pi Operating System

Figure 3 shows the screen of the robots operating system, Raspberry Pi. The functionality of the operating system is very similar to any other operating system we're used to, because Raspberry Pi is its own computer. NoMachine just allows us a way to wirelessly connect.

One of the major hurdles we had initially was getting used to using the command terminal in order to navigate the system and find what we needed to. We had very little knowledge of how to use the terminal, and even less knowledge of what commands we needed to use to get there. By combining the knowledge we gained in class, the demo lessons provided by HiWonder, and python command information online, we were eventually able to integrate the use of the terminal and the scripts we wanted to run.


NoMachine - ubuntu, Ubuntu 18.04.6 LTS
Terminal - ubuntu@ubuntu: ~
File Edit View Terminal Tabs Help

```
ubuntu@ubuntu:~$ Raymundo Rodriguez, August Halbert
```

Fig. 4. Armpi-Pro terminal

Figure 4 shows the terminal of the Armpi-Pro. This is where we can view all of our files, and what allows us to be able to execute our python script.

Connecting to the robot is one thing, but then we needed to figure out how and where to create an executable file in python, and what commands we needed to use in order to create the kind of movement we wanted. After scouring through the provided lessons and getting a deeper understanding of how these systems work, we were able to import the mecanum wheel chassis demo and forward/inverse kinematics demo onto the robot. From here, we were able to find where these demos were stored within the file manager of the Raspberry Pi, and create our own executable python file. The mecanum wheel chassis lesson taught us the proper commands to allow the robot to drift and move in a square, while the forward/inverse kinematics lesson gave us the proper commands to use inverse kinematics to manipulate the arm.

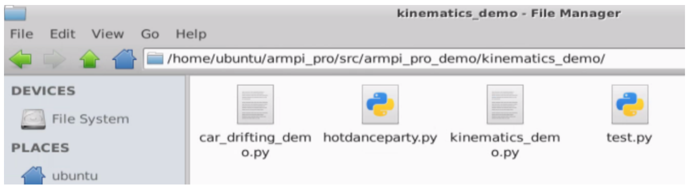


Fig. 5. ArmPi-Pro Source Files in Documents

In Figure 5 we show where some of the source files are. The file path within the file manager shows our script is located within the kinematics demo folder.

B. *Code Initialization*

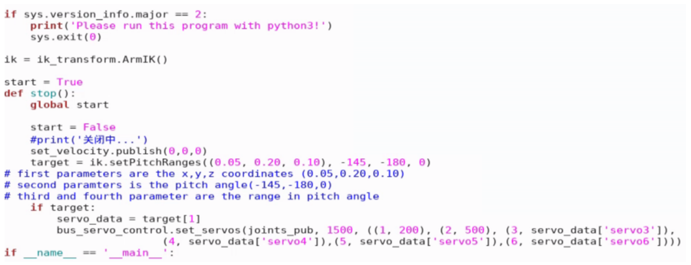
Once we found where the source files were located, we created our own executable python file and stored it in the same folder. We named our file **hotdanceparty.py**, and in this we started our project. The first thing we needed to do was import all of the important source files and functions that will allow us to initiate the system and use the libraries we need. The Armpi Pro comes with these functions and libraries built in, but we need to initiate them within the script or it won't be able to use those directories.

```
import sys
import time
import rospy
from kinematics import ik_transform
from armpi_pro import bus_servo_control
from chassis_control.msg import *
from hiwonder_servo_msgs.msg import MultiRawIdPosDur
```

Fig. 6. Imports Needed to Function

Figure 6 shares the import files necessary to make this program work. These include rospy, which as we discussed in class allows us to work with the robot operating system with python. From the kinematics library, we need to import the inverse kinematics transform function in order to have control of the arm. From the Armpi Pro library, we need to import servo and chassis control functions so that we hace control of the position of the servos across the robot.

After all of the imports are set up, we can begin to initialize our parameters and create a stop function that allows for it to shut down if anything is to start wrong.



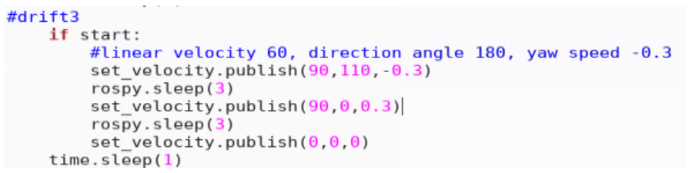
```python
if sys.version_info.major == 2:
    print('Please run this program with python3!')
    sys.exit(0)

ik = ik_transform.ArmIK()

start = True
def stop():
    global start

    start = False
    #print('关闭中...')
    set_velocity.publish(0,0,0)
    target = ik.setPitchRanges((0.05, 0.20, 0.10), -145, -180, 0)
# first parameters are the x,y,z coordinates (0.05,0.20,0.10)
# second paramters is the pitch angle(-145,-180,0)
# third and fourth parameter are the range in pitch angle
    if target:
        servo_data = target[1]
        bus_servo_control.set_servos(joints_pub, 1500, ((1, 200), (2, 500), (3, servo_data['servo3']),
                                (4, servo_data['servo4']),(5, servo_data['servo5']),(6, servo_data['servo6'])))
if __name__ == '__main__':
```

Fig. 7. Code Initialization

In Figure 7 we show the code that we need to start our main. The initial velocity is set to zero, and the target is defined. The target represents the position in which we want to maneuver the arm to reach. This is done using the inverse kinematics function setPitchRanges, which takes in the x, y, and z coordinates of the target, the pitch angle, and the range of the pitch angle as arguments. From this, we can set our initial target position. When hit, the servo data is extracted from the position. Using the set servos function, we are able to control the speed of movement, as well as the position of all of the servos.

C. Body and Arm Movement

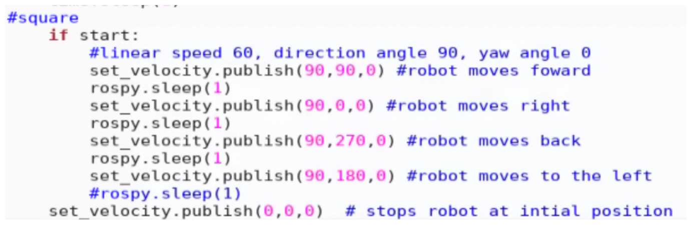
Now that everything has been properly initialized, we can start to input functions that allow the robot to perform its routine. In this routine, we have 4 different chassis moving sequences (3 drifting sequences and 1 square sequence), and 3 different arm moving sequences.



```python
#drift3
    if start:
        #linear velocity 60, direction angle 180, yaw speed -0.3
        set_velocity.publish(90,110,-0.3)
        rospy.sleep(3)
        set_velocity.publish(90,0,0.3)
        rospy.sleep(3)
        set_velocity.publish(0,0,0)
    time.sleep(1)
```

Fig. 8. Drift Sequence

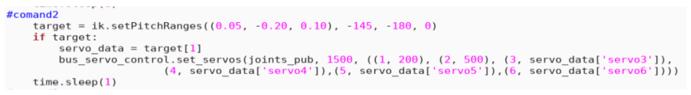
Figure 8 shows one of the drift sequences we created. Using the set velocity publish function, we were able to control the linear velocity, direction angle and yaw speed. The direction angle and yaw are what allow the robot to have the angular movement of the back wheels. The three functions we use set it to drift one direction, then the other, then come back to initial position. The rospy sleep function allows us to put small delays between the movements.



```python
#square
    if start:
        #linear speed 60, direction angle 90, yaw angle 0
        set_velocity.publish(90,90,0) #robot moves foward
        rospy.sleep(1)
        set_velocity.publish(90,0,0) #robot moves right
        rospy.sleep(1)
        set_velocity.publish(90,270,0) #robot moves back
        rospy.sleep(1)
        set_velocity.publish(90,180,0) #robot moves to the left
        #rospy.sleep(1)
    set_velocity.publish(0,0,0)  # stops robot at intial position
```

Fig. 9. Square Sequence

Figure 9 shows the square sequence we created. This uses the same functions and concepts as the drift sequence, but uses a different set of values in order to achieve the correct motion. For these functions, the direction angle was kept 90 degrees apart from each consecutive motion, in order to ensure the box formation. The yaw angle is 0 as well, to ensure we don't see any angular motion.



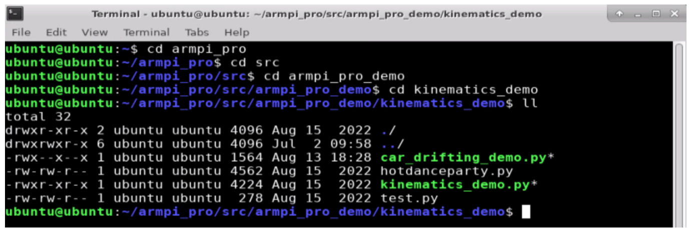
```python
#comand2
    target = ik.setPitchRanges((0.05, -0.20, 0.10), -145, -180, 0)
    if target:
        servo_data = target[1]
        bus_servo_control.set_servos(joints_pub, 1500, ((1, 200), (2, 500), (3, servo_data['servo3']),
                                (4, servo_data['servo4']),(5, servo_data['servo5']),(6, servo_data['servo6'])))
    time.sleep(1)
```

Fig. 10. Arm Swing Sequence

Figure 10 shows an example command that makes up an arm movement sequence. One command produces one movement, as it utilizes inverse kinematics in order to reach its destination. One sequence contains 5-8 commands in order to contain multiple motions, and we have 3 arm movement sequences in the routine. As with the initialization, we use the inverse kinematics function in order to redefine the target, and set the new target values. We can then use the servo control function in order to alter the servo positions. The first servo represents the gripper, which is normally open at the position 200, and closed at position 1000. The rest of the servos follow sequentially down the length of the arm.

D. Execution

In order to execute the script, we must find the correct file location to execute under using the command terminal. Using the change directory (cd) command, we can navigate through the files in the terminal.

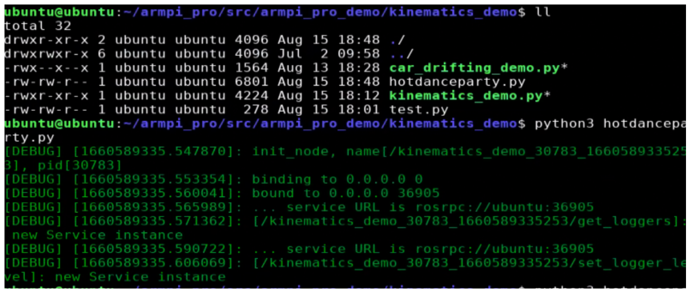

Terminal - ubuntu@ubuntu: ~/armpi_pro/src/armpi_pro_demo/kinematics_demo
File
Edit
View
Terminal
Tabs
Help

```
ubuntu@ubuntu:~$ cd armpi_pro
ubuntu@ubuntu:~/armpi_pro$ cd src
ubuntu@ubuntu:~/armpi_pro/src$ cd armpi_pro_demo
ubuntu@ubuntu:~/armpi_pro/src/armpi_pro_demo$ cd kinematics_demo
ubuntu@ubuntu:~/armpi_pro/src/armpi_pro_demo/kinematics_demo$ ll
total 32
drwxr-xr-x 2 ubuntu ubuntu 4096 Aug 15  2022 ./
drwxrwxr-x 6 ubuntu ubuntu 4096 Jul  2 09:58 ../
-rwx--x--x 1 ubuntu ubuntu 1564 Aug 13 18:28 car_drifting_demo.py*
-rw-rw-r-- 1 ubuntu ubuntu 4562 Aug 15  2022 hotdanceparty.py
-rwxr-xr-x 1 ubuntu ubuntu 4224 Aug 15  2022 kinematics_demo.py*
-rw-rw-r-- 1 ubuntu ubuntu  278 Aug 15  2022 test.py
ubuntu@ubuntu:~/armpi_pro/src/armpi_pro_demo/kinematics_demo$
```

Fig. 11. File Directory Pathway Using Terminal

Using the ll command, we are able to see all of the files within the folder our terminal is currently in. When under the kinematics demo folder, our script hotdanceparty.py is one of the listed files as shown in Figure 11.

In order to run the script, the command python3, followed by the name of the file to be executed can be used. Python3 needs to come first to ensure the script will run using python 3, otherwise it will not run. Then, hotdanceparty.py can be called since we are operating in the folder where the file is located.



```
ubuntu@ubuntu:~/armpi_pro/src/armpi_pro_demo/kinematics_demo$ ll
total 32
drwxr-xr-x 2 ubuntu ubuntu 4096 Aug 15 18:48 ./
drwxrwxr-x 6 ubuntu ubuntu 4096 Jul  2 09:58 ../
-rwx--x--x 1 ubuntu ubuntu 1564 Aug 13 18:28 car_drifting_demo.py*
-rw-rw-r-- 1 ubuntu ubuntu 6801 Aug 15 18:48 hotdanceparty.py
-rwxr-xr-x 1 ubuntu ubuntu 4224 Aug 15 18:12 kinematics_demo.py*
-rw-rw-r-- 1 ubuntu ubuntu  278 Aug 15 18:01 test.py
ubuntu@ubuntu:~/armpi_pro/src/armpi_pro_demo/kinematics_demo$ python3 hotdancepa
rty.py
[DEBUG] [1660589335.547870]: init_node, name[/kinematics_demo_30783_166058933525
3], pid[30783]
[DEBUG] [1660589335.553354]: binding to 0.0.0.0 0
[DEBUG] [1660589335.560041]: bound to 0.0.0.0 36905
[DEBUG] [1660589335.565989]: ... service URL is rosrpc://ubuntu:36905
[DEBUG] [1660589335.571362]: [/kinematics_demo_30783_1660589335253/get_loggers]:
 new Service instance
[DEBUG] [1660589335.590722]: ... service URL is rosrpc://ubuntu:36905
[DEBUG] [1660589335.606069]: [/kinematics_demo_30783_1660589335253/set_logger_le
vel]: new Service instance
```

Fig. 12. Executing the File

After entering the command, the debug process begins as shown in Figure 12. This uploads the selected script to the robot, and immediately begins running the program.

IV. Results

When this program is executed, the Armpi Pro completes a series of movement sequences, alternating between chassis movement and arm movement. There are 7 total sequences, and the entire routine lasts around 45 seconds. This project took days of trial and error, with a lot of supplemental learning in order to get to where we got. Initially, we had challenges trying to operate VirtualBox, and use NoMachine and Visual Studios inside of the virtual machine. Since this was where we were most comfortable after working with it in class, we thought this would be the easiest place to start. After being unable to install and run what we would need to, we moved onto other options. In a previous lab, we were able to connect to NoMachine using the robots wifi, and that was what we needed to be able to finally connect to the robot.

After finally connecting to NoMachine, we faced a large learning curve in order to utilize the Raspberry Pi the way we wanted to. We began by trying to use various github programs and python tutorials to try to figure out what we needed to do. Since we needed the specific commands for the Armpi Pro, this wasn't possible. Figuring out how to work with the terminal, how to navigate the source files and create our own, and how to integrate the two took a lot of trial and error. Using the demos provided by HiWonder, we were able to finally upload everything we needed to in order to work with the code. This code took a lot of working with and tweaking in order to determine what each of the parameters did, and the threshold values for each parameter.

V. Discussion of Results/Next Steps

This project allowed us to further understand core class concepts such as forward and inverse kinematics, python and command terminal navigation, virtual machines, and robotic operating systems (ROS). Inverse kinematics is essential for understanding how the robot calculates the motion of the arm. Understanding the frame and how the arm transverses to get to the target coordinates helped us to create the correct values to get the motion we desired. ROS was used within our script to communicate with the Raspberry Pi and ensure the routine ran smoothly. We encountered a lot of issues during this project, and the determination we had to continue trying to figure out what we wanted to do produced a project we are proud of.

Working with the Armpi Pro has been an insanely fun and informative project. We would like to be able to continue to work with these robots in the future, and understand more of the capabilities the robot offers. This project introduced us to working with Raspberry Pi and virtual machines, which we can use to create project of a similar, and more complex application.

References

[1] HiWonder. (2022). Armpi Pro. Google Drive. https://drive.google.com/drive/folders/1UNf2xl2Lc8w-Axk9q2BSJ8_6pFhUJMJ0?usp=drive_link