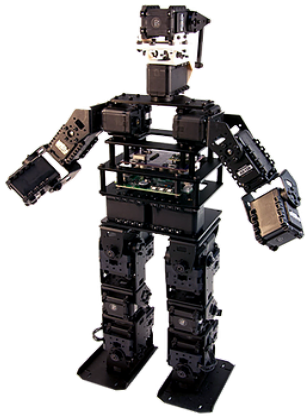
INTELLIGENT ROBOTICS- I



Final Project report

By

Dheerajchand Vummidi
Nauvin Ghorashian
Aditya Channamallu
Alvin Lin

Introduction and Project goals:

- Robot theater should be able to use computer vision to perform plays which follow a script
- Based on the initial positions of the Jimmys (and other obstacles on the stage) the play should vary slightly even when following the same script
- Multiple Jimmy's should be able to perform at the same time and not collide or play actions out of order
- Commands to the Jimmy must be received over WiFi
- Jimmys must be able to switch from walking mode to motion mode seamlessly
- The Jimmys must be able to move freely around the stage autonomously for at least ~15 minutes

Background:

HR-OS1 Framework includes several projects:

- Dxl_monitor
  - Displays dynamixel servo output data - ID, position, status
- RME
  - Create and playback series of moves that appear like fluid motion
- Node_server
  - A Javascript based web-GUI/API for high level functions such as RME motions and walking
- Walk_tuner
  - Controls a vast number of walking behavioral configurations, can be edit and save different profiles
- Ps3_demo
  - Allows the use of a PS3 bluetooth controller to make the robot walk around

Weekly progress:

Week1: Nothing much this week. Projects were assigned during the week 2.

Week2:

This week, project was assigned to us and we started working on it. But the idea

Wasn't much clear to us.So we approached meileh.Finally, we were fixed for the below objectives

- Receive data packets on the Jimmy RasPi via WiFi (Python Script)
- Determine how to use the HR-OS1 software framework to make the Jimmy robot perform actions
- Integrate parts 1 and 2 to receive, decode, and execute commands on the Jimmy robot received from the other teams

We didn't recieve Jimmy yet. This week.

Week3:

This week , we got a new idea to control the motion of the robot by giving automated inputs to the RMe instead of hacking the framework.That time we didnt realise that RMe doesn't support walking functionality,We were still waiting for the Jimmy this week.

Week4:

This week we were mostly into doing Hw1 and making it successful.Atlast we got the jimmy into our hands. We tried searching for winRme on the Jimmy but couldnt find it on the installed copy of HR-os 1. So we did a complete fresh install from Github and got the winRME functional on the robot.

Week5:

This week we started experimenting with WinRme and tested various motions.These were the major concerns for that week.

Raspberry Pi randomly resetting -- We are encountering an issue with the Jimmy Robot where the Pi is frequently resetting. We are observing this issue while using the RME tool to tell the bot to perform actions, and halfway through an action it resets and the servos immediately stop moving. This seems to happen at random although some specific commands seem to cause the restart more than others.

Motor 16 needs to be fixed to fully test the walking function.

Week6:

- Analyzed the code and thought about how we can control the robot with existing functions.
- Used PS3 controller to make the robot walk.
- Found one link function between PS3 joystick and robot walking
- Was able to make the robot walk in a straight line but not turning left or right.

Week7:

- wrote a program which make the robot play an input motion. (e.g. ./motion sit)
- wrote a program which make the robot turn and walk forward.
- discussed with team 2 about the format of the commands

Week8:

- Written the script required to walk the robot in single direction
- Written the script required to play the requested robotic actions.
- Started writing final project report

Implementation

At first, we looked into using the node_server project to perform the necessary functions we needed for our project. On the surface level, this application seemed to present a suitable API into the inner workings of the HR-OS1 framework. This would allow us to focus our

efforts on integrating Team 1's actions and Team 2's pathfinding algorithm. However, we discovered that the node_server project was incomplete, and our attempts to execute API calls through this program were unsuccessful. Since none of our team members is proficient in Javascript, we decided not to dedicate any time to attempt to make this work for our project.

We spoke with Melih about our options and he kindly arranged an introduction for our team with Dan Petre, a former PhD student who has some familiarity with the HR-OS1 framework. He showed us the code he had worked on which provides an API to allow various high-level functions to be called, such as walking, turning, and playing actions. He had not yet fully implemented some of the necessary parts of his API to satisfy the needs of our project, and suggested that we could pick up where he left off to complete the code. However, he did all of his development using an Intel Edison board and a complex build environment, which he estimated would take roughly 3 weeks to set up (given the amount of time we had available each week to contribute to the project), before we could even start development. This challenge ultimately led us to decide to emulate his strategy of implementation while remaining in our original development environment on the Raspberry Pi.

To emulate Dan's framework, we used two key existing projects in the HR-OS1 code base: RME and Walk_tuner. RME (Robot Motion Editor) is a tool which allows a user to program and playback a series of poses on the Jimmy, which demonstrates an animated action. Walk_tuner is a configuration tool which allows a user to modify several settings pertaining to the walking posture and gait of the Jimmy. The Walk_tuner GUI presents all of the different parameters to

the user, which can be changed in real time. A walk on/off toggle option allows the user to see how the modifications affect the walking behavior of the robot. To make effective use of this existing program, we copied the full source code of Walk_tuner and then kept only the necessary functions to emulate this environment. We also eliminated the GUI portion, as we knew we would be creating our own GUI, and replaced the ability to modify the walking parameters with two function calls: walk and turn. Each of these two functions contained pre-set values to modify the walking parameters which in turn allowed the Jimmy to either walk or turn when the functions were called. Based on arguments passed through the terminal on our newly programmed GUI (shown below), the Jimmy would perform the desired action.

Lastly, we also needed to allow the user to command the Jimmy to perform a range of pre-defined motions. For this task, we turned to the functionality built into the RME tool. Similarly, we copied the parts of the code necessary to load configuration files and play them back, and implemented this as a function with similar arguments to the previous two.

Program GUI:

```
********** Walking and Motion Manager **********

    Input Format: 2 arguments
    The second argument:
        1. walk:   number of seconds (+/forward -/backward)
        2. turn:   degree (+/turn left -/turn right)
        3. motion: page_number

    Examples:
        1   5 for walking forward 5 seconds
        2 -45 for turning right by 45 degrees
        3   4 for playing page #4

    Input:
```

Command Format:

1. Walk: the second argument is the time in second Jimmy will walk. If the value is positive, Jimmy walks forward for the specified number of seconds. If the value is negative, Jimmy walks backward for the absolute value of the input in seconds. If the previous motion played is not walk, turn, or walk ready, walk ready will be automatically played prior to walking.
2. Turn: the second argument is how many degrees Jimmy turns. For a right turn, the input should be positive. For a left turn, the input should be negative. If the previous motion played is not walk, turn, or walk ready, walk ready will be played prior to turning.
3. Play Motion: the second argument is the page number of a motion

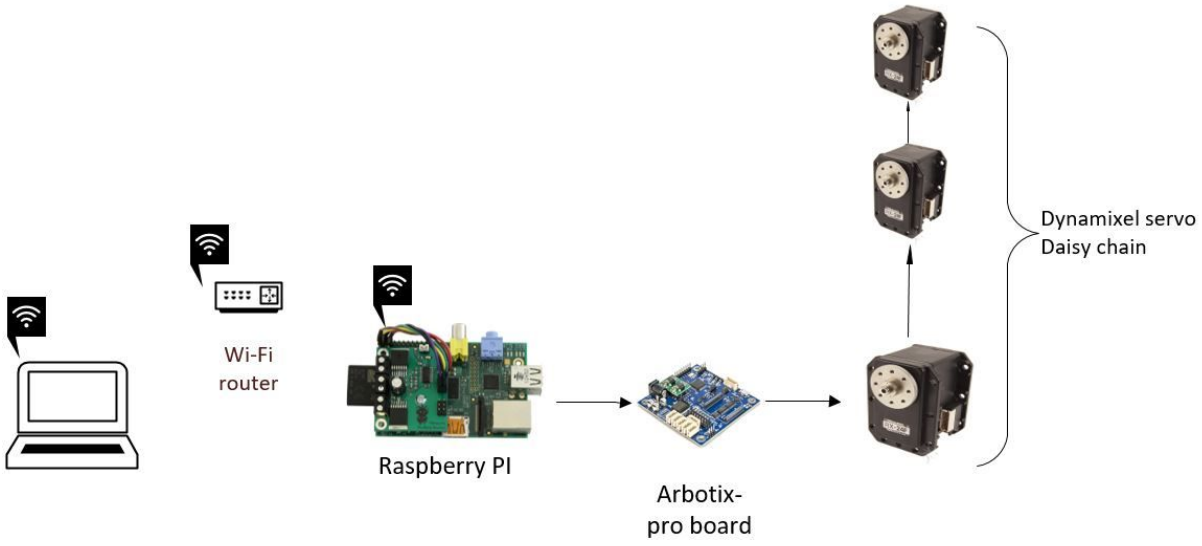
Functions:

Three functions are to control walking, turning, and playing motions.

- walk(direction, seconds)
  - direction == 1, walk forward
  - direction == -1, walk backward
- turn(direction, degrees)
  - direction == 1, turn left
  - direction == -1, turn right
  - turn in place
- motion(page_number)

These functions can be used directly in the theater script if the required time of each motion is known

Block diagram:



We can observe the following points from the above image.

1. Commands are sent from a laptop connected to a wifi network
2. A raspberry pi which is on Jimmy will receive those commands. It is also connected to the same wifi network.
3. Now the Raspberry pi will command the Arbotix pro board and this board is the commander for all the 20 servos on the robot.
4. Now the commands are sent to all the motors as they are daisy chained to eachother.

Testing:

We had to fully disassemble Jimmy twice for replacing the cables.Below are the image showing that.

Finally, it started working and below link provides a quick video of our work on Jimmy. The video includes jimmy movements, walking and various poses from the RME.

fully functional video- https://youtu.be/GSPSkpZoVLo

Limitation:

We ran into some problems trying to integrate the customized walk_tuner function and the ability to play RME motions. Initially, we found that much of the code between the two libraries is replicated, especially some of the code that governs how each program

Nextsteps:

- Add functionality to stream in commands from external source instead of requiring human interface
- Optimize code to reduce number of initialization steps and speed up execution
- Make the robot turn at different speeds & turn while walking
- Delve deeper into framework code to further isolate necessary functions from walk_tuner and RME

Problems faced their solutions:

1. Motors overheat after 15 minutes of continuous usage - don't operate motors continuously
2. Automatic power down on Lipo batteries -- use AC power supply instead.
3. Not good traction for walking on normal floor. -- use carpet floor instead. Or attach rubber pads to the legs.
4. Broken servo

a. Replace servo
b. Replaced all the connecting cables

5. Unable to assign servo IDs after replacement
    a. Acquire new Jimmy
    b. New robot came with edison board instead of Raspberry Pi

6. Start a program when each command is received takes time to initialize

    a. Put everything in one program and wait for inputs

7. Make Jimmy walk or turn before it's in walk ready position results in falling
    a. Continuously track the page number being played
    b. If it is not walk_rdy, play it before walking or turning

Source Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <termios.h>
#include <term.h>
#include <ncurses.h>
#include <signal.h>
#include <libgen.h>
#include "cmd_process.h"
#include "mjpg_streamer.h"
#include <iostream>
#define MOTION_FILE_PATH        "../../../Data/motion_4096.bin"
```

```
#define INI_FILE_PATH       "../../../Data/config.ini"
using namespace std;
using namespace Robot;
LinuxMotionTimer linuxMotionTimer;
LinuxArbotixPro linux_arbotixpro("/dev/ttyUSB0");
ArbotixPro arbotixpro(&linux_arbotixpro);
minIni* ini;
void walk(int direction, int second){
        printf("walking...");
        Walking::GetInstance()->LoadINISettings(ini);
        linuxMotionTimer.Start();

MotionManager::GetInstance()->AddModule((MotionModule*)Walking::GetIn
stance());
        MotionManager::GetInstance()->SetEnable(true);
        MotionManager::GetInstance()->ResetGyroCalibration();
        Walking::GetInstance()->X_OFFSET = direction * -2;
        Walking::GetInstance()->A_MOVE_AMPLITUDE = 0;
        Walking::GetInstance()->X_MOVE_AMPLITUDE = direction * 10;
        Walking::GetInstance()->Start();
        usleep(1000000 * second);
        Walking::GetInstance()->Stop();
        Walking::GetInstance()->X_MOVE_AMPLITUDE = 0;
        Walking::GetInstance()->Y_MOVE_AMPLITUDE = 0;
        Walking::GetInstance()->A_MOVE_AMPLITUDE = 0;
        usleep(1000000);
        MotionManager::GetInstance()->SetEnable(false);

MotionManager::GetInstance()->RemoveModule((MotionModule*)Walking::
GetInstance());
        linuxMotionTimer.Stop();
        printf(" done\n");
}
```

```
void turn(int direction, int degree){
	printf("turning\n");
	int turn_degree;
	if(degree < 90)
	turn_degree = 23;
	else
	turn_degree = 45;
	int sec = degree/(turn_degree*2.0) * 4.0;
	Walking::GetInstance()->LoadINISettings(ini);
	linuxMotionTimer.Start();

MotionManager::GetInstance()->AddModule((MotionModule*)Walking::GetIn
stance());
	MotionManager::GetInstance()->SetEnable(true);
	MotionManager::GetInstance()->ResetGyroCalibration();
	printf("turning...\n");
	Walking::GetInstance()->X_OFFSET = -4;
	Walking::GetInstance()->Y_OFFSET += 5;
	Walking::GetInstance()->A_MOVE_AMPLITUDE = turn_degree *
direction;
	Walking::GetInstance()->X_MOVE_AMPLITUDE = 0;
	Walking::GetInstance()->Start();
	usleep(1000000 * sec);
	Walking::GetInstance()->Stop();
	Walking::GetInstance()->X_MOVE_AMPLITUDE = 0;
	Walking::GetInstance()->Y_MOVE_AMPLITUDE = 0;
	Walking::GetInstance()->A_MOVE_AMPLITUDE = 0;
	usleep(1000000);
	MotionManager::GetInstance()->SetEnable(false);

MotionManager::GetInstance()->RemoveModule((MotionModule*)Walking::
GetInstance());
	linuxMotionTimer.Stop();
```

```
        printf(" Done\n");
}
void motion(int page_num){
        printf("playing ");
        MotionManager::GetInstance()->LoadINISettings(ini);
        MotionManager::GetInstance()->SetEnable(false);

MotionManager::GetInstance()->AddModule((MotionModule*)Action::GetInst
ance());
        linuxMotionTimer.Stop();
        PlayCmd(&arbotixpro, page_num);

MotionManager::GetInstance()->RemoveModule((MotionModule*)Action::G
etInstance());
}
void change_current_dir()
{
        char exepath[1024] = {0};
        if (readlink("/proc/self/exe", exepath, sizeof(exepath)) != -1)
        chdir(dirname(exepath));
}
void sighandler(int sig)
{
        struct termios term;
        tcgetattr( STDIN_FILENO, &term );
        term.c_lflag |= ICANON | ECHO;
        tcsetattr( STDIN_FILENO, TCSANOW, &term );
        exit(0);
}
int main(int argc, char *argv[])
{
        signal(SIGABRT, &sighandler);
        signal(SIGTERM, &sighandler);
```

```
        signal(SIGQUIT, &sighandler);
        signal(SIGINT, &sighandler);

        ini = new minIni(INI_FILE_PATH);
        change_current_dir();

        if(Action::GetInstance()->LoadFile(MOTION_FILE_PATH))
printf("Motions Loaded\n");
        //mjpg_streamer* streamer = new mjpg_streamer(0, 0);
        //httpd::ini = ini;
        ////////////////////// Framework Initialize ////////////////////////////
        if (MotionManager::GetInstance()->Initialize(&arbotixpro) == false)
        {
        printf("Fail to initialize Motion Manager!\n");
        return 0;
        }

        bool on = TRUE;
        int  input1;
        int  input2;
        int  sec = 0;
        int  sign;
        int  degree;
        int  prev_page = 0;

        while(on){
 system("clear");
 cout << "******************** Walking and Motion Manager
********************\n\n";
 cout << "   Input Format: 2 arguments\n";
 cout << "   The second argument:\n";
 cout << "           1. walk:   number of seconds (+/forward -/backward)\n";
 cout << "           2. turn:   degree (+/turn left -/turn right)\n";
```

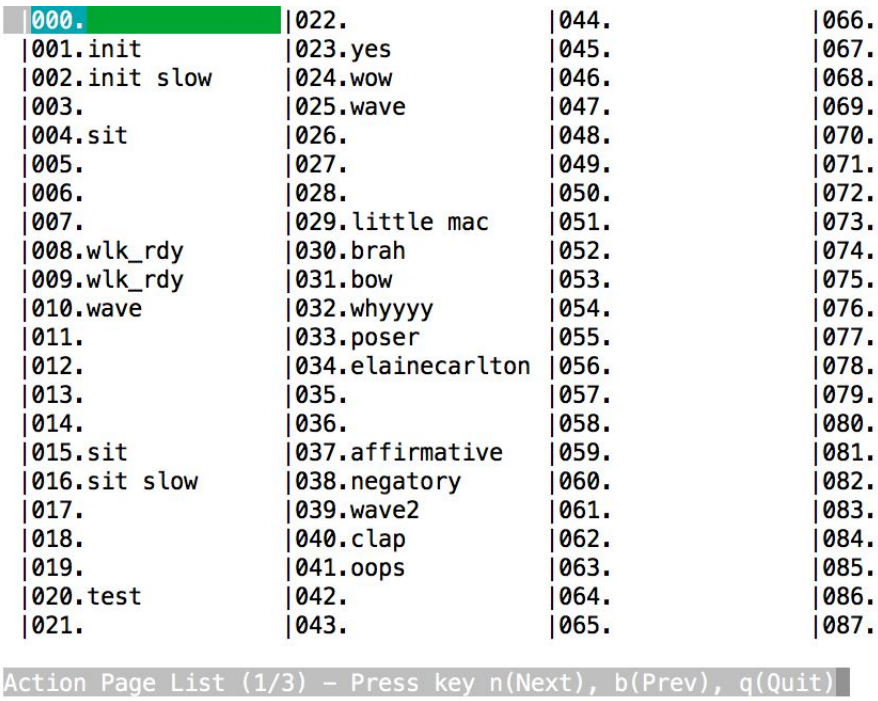
```
cout << "            3. motion: page_number\n\n";
        cout << "    Examples:\n";
cout << "            1   5 for walking forward 5 seconds\n";
        cout << "    2 -45 for turning right by 45 degrees\n";
        cout << "    3   4 for playing page #4\n\n";
cout << "   Input: ";
        cin >> input1 >> input2;
linuxMotionTimer.Initialize(MotionManager::GetInstance());
cout << "   Status: ";
        switch(input1){
        case 1:
 if(prev_page != 8){
  prev_page = 8;
  motion(8);
 }
 if(input2 < 0)
  sign = -1;
 else
  sign = 1;
                sec = sign * input2;
 walk(sign, sec);
                break;
        case 2:
 if(prev_page != 8){
  prev_page = 8;
  motion(8);
 }
 if(input2 < 0)
  sign = -1;
 else
  sign = 1;
 if(input2 * sign < 90)
  degree = 23;
```

```
  else
   degree = 45;
  turn(sign, sign * input2);
        break;
        case 3:
  prev_page = input2;
  motion(input2);
  break;
        default:
  printf("invalid input\n");
  usleep(1500000);
        }
        }
        exit(0);
}
```

Screenshots:

Rme:

This is a powerful tool and it can be used to create a new motion for the robot or to we can even use the pre programmed motions in this library.It is as shown below.



DXL monitor:

This is another tool which can aid us in the case of Servo device ID loss. Every servo motor will be assigned an ID number to it. When they lose that number we can use this tool to re initialise the ID numbers.

```
[Dynamixel Monitor for DARwIn v1.00]

 Check ID:1(R_SHOULDER_PITCH)     ... OK
 Check ID:2(L_SHOULDER_PITCH)     ... OK
 Check ID:3(R_SHOULDER_ROLL)      ... OK
 Check ID:4(L_SHOULDER_ROLL)      ... OK
 Check ID:5(R_ELBOW)              ... OK
 Check ID:6(L_ELBOW)              ... OK
 Check ID:7(R_HIP_YAW)            ... OK
 Check ID:8(L_HIP_YAW)            ... OK
 Check ID:9(R_HIP_ROLL)           ... OK
 Check ID:10(L_HIP_ROLL)          ... OK
 Check ID:11(R_HIP_PITCH)         ... OK
 Check ID:12(L_HIP_PITCH)         ... OK
 Check ID:13(R_KNEE)              ... OK
 Check ID:14(L_KNEE)              ... OK
 Check ID:15(R_ANKLE_PITCH)       ... OK
 Check ID:16(L_ANKLE_PITCH)       ... OK
 Check ID:17(R_ANKLE_ROLL)        ... OK
 Check ID:18(L_ANKLE_ROLL)        ... OK
 Check ID:19(HEAD_PAN)            ... OK
 Check ID:20(HEAD_TILT)           ... OK
 Check ID:200(ARBOTIX_PRO)        ... OK

[ID:200(ARBOTIX_PRO)]
```

Walktuner:

This is a very helpful tool.We can understand about robot walking behaviour using this tool.We can even create customized walking styles by tweaking the settings in this menu.

Walking Mode(on/off)	OFF
X offset(mm)	5
Y offset(mm)	20
Z offset(mm)	20
Roll(x) offset(degree)	0.0
Pitch(y) offset(degree)	-10.0
Yaw(z) offset(degree)	0.0
Hip pitch offset(degree)	3.6
Auto balance(on/off)	ON
Period time(msec)	907
DSP ratio	0.32
Step forward/back ratio	0.25
Step forward/back(mm)	0
Step right/left(mm)	0
Step direction(degree)	0
Turning aim(on/off)	OFF
Foot height(mm)	30
Swing right/left(mm)	25.0
Swing top/down(mm)	2
Pelvis offset(degree)	0.1
Arm swing gain	1.8
Balance knee gain	0.07
Balance ankle pitch gain	0.07
Balance hip roll gain	0.07
Balance ankle roll gain	0.07
BalanceAngleGain	0.07
BalanceAngleSmoothGain	0.95
Lean FB Accel	0.00

]

References:

Provided by Mathias:

- Repository for HR-OS1: https://github.com/Interbotix/HROS1-Framework
- Repository for controlling Jimmy from ROS: https://github.com/msunardi/arbotix_to_pololu
- Repository for the ROS package for REBeL: https://github.com/msunardi/rebel_ros
- HR-OS1 tutorial for Robot Motion Editor (RME): https://www.youtube.com/watch?v=lOt36Otpp4Y

Pairing PS3 controller: https://github.com/Interbotix/HROS1-Framework/wiki/ps3_demo#pairing-the-sixaxis-controller-to-your-robot