Dancing Hexapod

Vivan Bhalla, Sam Burkhart

PSU ECE 578

Fall 2017

Final Project

Contents

[Goal: 2](#_Toc500882352)

[Design: 2](#_Toc500882353)

[Hexapods: 2](#_Toc500882354)

[Mechanical Design: 2](#_Toc500882355)

[Electrical Design: 4](#_Toc500882356)

[Software: 6](#_Toc500882357)

[Stage Manager: 15](#_Toc500882358)

[Results: 17](#_Toc500882359)

[Conclusion: 17](#_Toc500882360)

[References: 18](#_Toc500882361)

[Appendices: 18](#_Toc500882362)

[18DOF Hexapod BOM: 18](#_Toc500882363)

[12DOF Hexapod BOM: 18](#_Toc500882364)

# Goal:

Design a stage system where a host computer acts as a “stage manager”, orchestrating a hexapod “actors” and other stage components (speakers, spotlights, etc.). Demonstrate that a hexapod can be directed to follow a spotlight and perform dancing behaviors. This broke down into two major tasks, designing the hexapod actors that can be controlled to move and dance, and designing a stage which can take a global camera view as input, identify the hexapod and the spotlight, and command the hexapod to move under the spotlight.

For designing the hexapod “actors”, one 12 degree-of-freedom (DOF) hexapod was to be resurrected from last year’s project, while one 18 DOF hexapod was designed new this year. Sam worked on the hexapod design.

The stage involved a webcam positioned above the stage connected to a laptop. The host laptop utilized OpenCV to identify the hexapod’s position and spotlights position, then determine the correct commands to move the hexapod under the spotlight and dance. The OpenCV code is based on Vivan’s work.

# Design:

## Hexapods:

A hexapod is a 6 legged robot. The two hexapods designed for this project are a 12 DOF version, where there are 2 servos per leg (one for rotation and one for lift), and an 18 DOF version, where there are 3 servos per leg (one for rotation, one for the upper leg, and one for the lower leg).

### Mechanical Design:

#### 12DOF Hexapod:

The 12DOF hexapod chassis and servos were already assembled. The chassis is yellow and designed with plastic sheeting. The body is rectangular with legs spaced evenly on each side of the rectangle.



Figure 1: 12DOF Yellow Hexapod

The angle movement of the legs is limited to a perpendicular angle to the body in the rear and about 30 degrees to the front. The main body is created by two parallel plates, with a two-servo rotating board sandwiched between each “knob”. The legs are created using three beams of plastic. This limits the overall range of motion of the leg, but allows for greater stability by reducing the pressure on the servos as the hexapod has a neutral standing position.



Figure 2: Leg Diagram

The symmetric design of the 12DOF yellow robot allows for gestures to be programmed very easily as each leg has the same front/back and up/down range of motion.

#### 18DOF Hexapod:



Figure 3: SainSmart 18DOF Hexapod

The chassis of the 18DOF hexapod chassis was purchased by SainSmart, and assembled following the instructions listed here: <http://repository.sainsmart.com/index.php?share/file&user=bbu_sainsmart&sid=boPzWnBK>

The servo rotation brackets are connected to the servo output shaft on the top and a metal bearing on the bottom. The center legs are slightly outside the rectangle created by the 4 front and back rotational servo mounts. The rear leg rotations are limited by the rotational servo frames. The front legs do not have this limitation and are able to move forward allowing front legs to reach a parallel position. The lower legs (closer to the hexapod frame) can rotate 180 degrees from vertically upright, to fully downward. Similarly the upper legs (furthest from the hexapod frame) can retract close to the body (as in figure 3) to fully extended from the lower leg, roughly 180 degrees. This flexibility allows for greater programming options, but also puts more strain on each individual joints.



Figure 4: MG996R Servo

The servos are those recommended within that tutorial: MG996R Digital Metal Gear 10kg servos purchased from Kuman. The specification sheet is located here: <http://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf>

The MG966R weighs 55 grams, has a speed of 0.98 – 1.2 rpm (at 4.8V-6V) and a 180 degree range of motion +-90 degrees, and a stall torque of 9.4 – 11 kgf\*cm (4.8V-6V), a running current of 500-900 mA, a stall current of 2.5A at 6V, and an input voltage of 4.8V-7.2V. It has a deadband width of 5 microseconds. It uses a 50Hz/20ms PWM signal.

### Electrical Design:

There are two major electrical components to the each hexapods, the servo controller board and the main compute board. The compute board chosen was the RaspberryPi 3, due to its low price, ease of programming, and good community support. The servo board chosen is the Adafruit 16-channel 12-bit PWM Servo PiHat, for its good community support and ease of programming.

#### Compute Board:

The RaspberryPi 3 details can be found here: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>

The power for the RaspberryPi was provided by a 5V USB wall adapter. Communication to the RaspberryPi is performed through WIFI, via an SSH connection for maintenance and debug, Jupyter Notebook environment for development, and a python client/server socketed connection during action.

#### Servo Controller Board:

The details of the servo controller board can be found here: <https://www.adafruit.com/product/2327>

The board is sold as a kit that must be soldered together. The 12DOF hexapod could use a single servo controller board with all of the pieces contained in a single kit. Adafruit provides a tutorial on how to setup and test the servo controller board here: <https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/>

The 18DOF hexapod required two servo controller boards. This requires special right-angle servo headers and tall header pins for the lower controller board. These are available at Adafruit as well. 9 servos are connected to the lower controller board and 9 are connected to the upper controller board. The servo controller board communicates with the RaspberryPi via I2C protocol. There are address pads to solder in order to change the address of each controller board allowing multiple boards to be stacked. For the 18DOF hexapod the lower controller board is assigned the default I2C address (0x40) and the upper controller board is assigned address 0x41. The right legs are controlled by the top controller boards, while the left legs are controlled by the bottom controller board.

The controller boards require a separate 5-6V power source per controller to power the servos. Each controller board has a terminal power connector and a 5x21mm DC barrel jack. The controller boards are currently powered by a USB wall adapter supplying 2.1A @5V and USB to 5x21mm barrel jack cables. This is insufficient for the current draw of the stall torque of the MG966R servos (2A per servo), causing instability during extreme motions. Ideally a larger power supply would be used, converting the higher voltage down to 6V, using a Lithium Ion battery allowing autonomous movement.

The servos are attached using [rotation, raise] order for the 12DOF hexapod, and [rotation, lower, upper] order for the 18DOF hexapods, as shown below. The exception is the 18DOF right\_front\_lower [channel 3] and right\_front\_rotate [channel 4].

|  |  |  |
| --- | --- | --- |
| **Controller Board Address** | **Channel Number** | **Servo Connected** |
| 0x40 | 0 | left\_back\_rotate |
| 0x40 | 1 | left\_back\_raise |
| 0x40 | 2 | right\_back\_rotate |
| 0x40 | 3 | right\_back\_raise |
| 0x40 | 4 | left\_center\_rotate |
| 0x40 | 5 | left\_center\_raise |
| 0x40 | 6 | right\_center\_rotate |
| 0x40 | 7 | right\_center\_raise |
| 0x40 | 8 | left\_front\_rotate |
| 0x40 | 9 | left\_front\_raise |
| 0x40 | 10 | right\_front\_rotate |
| 0x40 | 11 | right\_front\_raise |
| 0x40 | 12 | unused |
| 0x40 | 13 | unused |
| 0x40 | 14 | unused |
| 0x40 | 15 | unused |

Figure 5: 12DOF servo hookup mapping

|  |  |  |
| --- | --- | --- |
| **Controller Board Address** | **Channel Number** | **Servo Connected** |
| 0x40 | 0 | left\_front\_rotate |
| 0x40 | 1 | left\_front\_lower |
| 0x40 | 2 | left\_front\_upper |
| 0x40 | 3 | left\_center\_rotate |
| 0x40 | 4 | left\_center\_lower |
| 0x40 | 5 | left\_center\_upper |
| 0x40 | 6 | left\_back\_rotate |
| 0x40 | 7 | left\_back\_lower |
| 0x40 | 8 | left\_back\_upper |
| 0x40 | 9 | unused |
| 0x40 | 10 | unused |
| 0x40 | 11 | unused |
| 0x40 | 12 | unused |
| 0x40 | 13 | unused |
| 0x40 | 14 | unused |
| 0x40 | 15 | unused |
| 0x41 | 0 | right\_front\_rotate |
| 0x41 | 1 | right\_front\_lower |
| 0x41 | 2 | right\_front\_upper |
| 0x41 | 3 | right\_center\_lower |
| 0x41 | 4 | right\_center\_rotate |
| 0x41 | 5 | right\_center\_upper |
| 0x41 | 6 | right\_back\_rotate |
| 0x41 | 7 | right\_back\_lower |
| 0x41 | 8 | right\_back\_upper |
| 0x41 | 9 | unused |
| 0x41 | 10 | unused |
| 0x41 | 11 | unused |
| 0x41 | 12 | unused |
| 0x41 | 13 | unused |
| 0x41 | 14 | unused |
| 0x41 | 15 | unused |

Figure 6: 18DOF servo hookup mapping

### Software:

The software of the hexapod designs are broken into the operating system, servo controller board library, design notebooks, configuration yaml, hexapod module, server interface module, and client module.

#### Operating System:

The RaspberryPi is compatible with several operating systems, but by far the most common is Raspbian Stretch available here: <https://www.raspberrypi.org/downloads/raspbian/>

#### Controller Board Library:

The Adafruit servo controller board includes a Python library for interfacing with the servos. The library is available here: <https://github.com/adafruit/Adafruit_Python_PCA9685>

The code can be installed in your current python environment. For this project I utilized Python3 for all of the software. The Adafruit servo controller board tutorial mentioned in the electrical section explains how to test the installation of the servo controller board. You can import the library and instantiate the controller board object using code shown below:

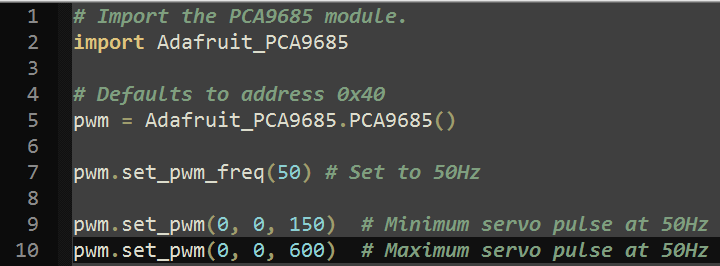


Figure 7: Servo controller board example

The pwm frequency selected is 50Hz (20ms period), with a minimum pulse of 150, and a maximum pulse of 600. The two major functions utilized to communicate to the servos are set\_pwm\_freq() to set the pwm frequency and set\_pwm() to set the PWM pulse. The first argument specifies the channel (0-15), the second argument is the on pulse, and the final argument is the off pulse. The on pulse is set to 0, allowing the off pulse to be set to the appropriate duty cycle to position the servo at +-90 degrees.

#### Jupyter Notebooks:

The development and experimentation of the hexapod movements were performed within a Jupyter notebook. Jupyter notebooks allow code to be organized into cells that can be executed individually and reorganized as necessary. More details on Jupyter notebooks are available here: <https://jupyter-notebook.readthedocs.io/en/stable/>

Jupyter can be installed using pip like “pip install jupyter”, and launched using a command like “jupyter notebook”.

Jupyter launches a server that can be accessed using a chrome web browser with the provided link displayed on the terminal after launching the notebook. There are two example notebooks that demonstrate how to use the hexapods within the project github in “hexapod/Hexapod Test 12DOF.ipynb” and “hexapod/Hexapod Test 18DOF.ipynb”.

#### Configuration YAML:

YAML format is a superset of JSON used for readability and ease of parsing, commonly used for configuration information. Using YAML, the configuration can be read directly into a Python object. This makes it relatively simple to configure the servo controller boards. In each configuration file there is a list of boards, where each board contains:

* a board\_address
* pwm\_freq
* list of servos

Each servo includes:

* name of the servo
* channel on the controller board the servo is connected to
* minimum range of the servo
* maximum range of the servo
* forward percentage of rotation
* back percentage of rotation
* up percentage of rotation
* down percentage of rotation
* center percentage of rotation
* invert the 0-100 percent range for left/right mirroring.

The rotation servos specify a 0/100% forward/back rotation using the invert field to ensure left and right rotations follow this definition of rotation. The 0/100% for up/down rotation follows a similar pattern for raise servos for the 12DOF hexapod and the lower/upper servos for the 18DOF hexapod. The center percentage is used to define a center aligned position, moving all of the legs to 90 degrees from the center-line of the hexapod. This is the input to the hexapod class, used to initialize the hexapods.

There are two configuration yamls, one for the 12DOF hexapod located in the github repository in “hexapod/config\_12DOF.yaml”. The 18DOF hexapod is located in “hexapod/config\_18DOF.yam

#### Hexapod\_module:

The hexapod.py module located in the github repository in “hexapod/hexpod.py” contains two base classes, Servo and Hexapod, and two sub-classes of Hexapod, Hexapod\_12DOF and Hexapod\_18DOF. The Servo class stores the key servo parameters listed in the configuration YAML, as well as the methods used to move the servos. Those functions are:

* **set\_max():** changes the maximum servo range
* **set\_min():** changes the minimum servo range
* **servo\_percent\_to\_pulse():** helper function to convert a 0-100% range to the appropriate servo scale based on the min/max values from the configuration yaml or modified by the set\_max()/set\_min() methods
* **set\_position():** changes the servo position based on a 0-100% range based on the servo’s min/max range values
* **set\_pulse():** changes the servo position based on a pulse value, used to help determine the min/max pulse for a given servo.
* **set\_center():** changes the center value in percent, used in the move\_center() method
* **set\_forward():** changes the forward value in percent, used in the move\_forward() method
* **set\_back():** change the back value in percent, used in the move\_back() method
* **set\_up():** change the up value in percent, used in the move\_up() method
* **set\_down():** change the down value in percent, used in the move\_down() method
* **move\_forward():** move the servo to the value specified in the forward field if specified
* **move\_back()**: move the servo to the value specified in the back field if specified
* **move\_up():** move the servo to the value specified in the up field if specified
* **move\_down():** move the servo to the value specified in the down field if specified
* **move\_center():** move the servo to the value specified in the center field if specified

The Hexapod class reads in the configuration object stored in the yaml files and instantiates the servos associated with the hexapod.

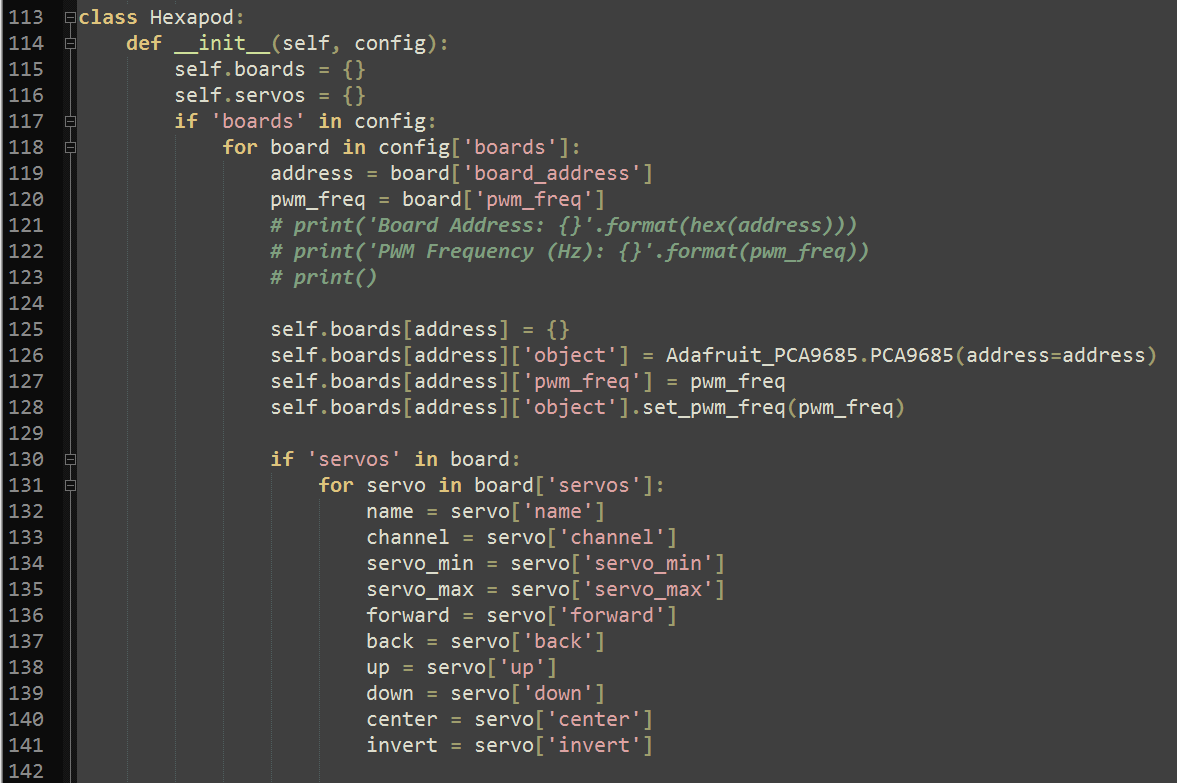


Figure 8: Hexapod constructor (first half)

The constructor for the Hexapod class iterates over the configuration object provided as an argument. It stores the board information in a boards dictionary keyed off of the boards I2C address. Each board is a dictionary with an ‘object’, ‘pwm\_freq’, ‘board\_address’, and ‘servos’ keys. The ‘object’ key points to the Adafruit controller board object used for sending commands to the servos. The ‘servos’ key points to a dictionary keyed off of the servos name. Each servo name key points to a Servo object, utilizing the ‘name’, ‘board’, ‘servo\_min’, ‘servo\_max’, ‘forward’, ‘back’, ‘up’, ‘down’, ‘center’, and ‘invert’ to invoke the Servo constructor. This allows the hexapod object to access each servo by name and use the helper functions (move\_forward, move\_back, move\_up, move\_down, move\_center) as well as the set\_position(<%>) and set\_pulse(<pulse>) functions to position the servos.

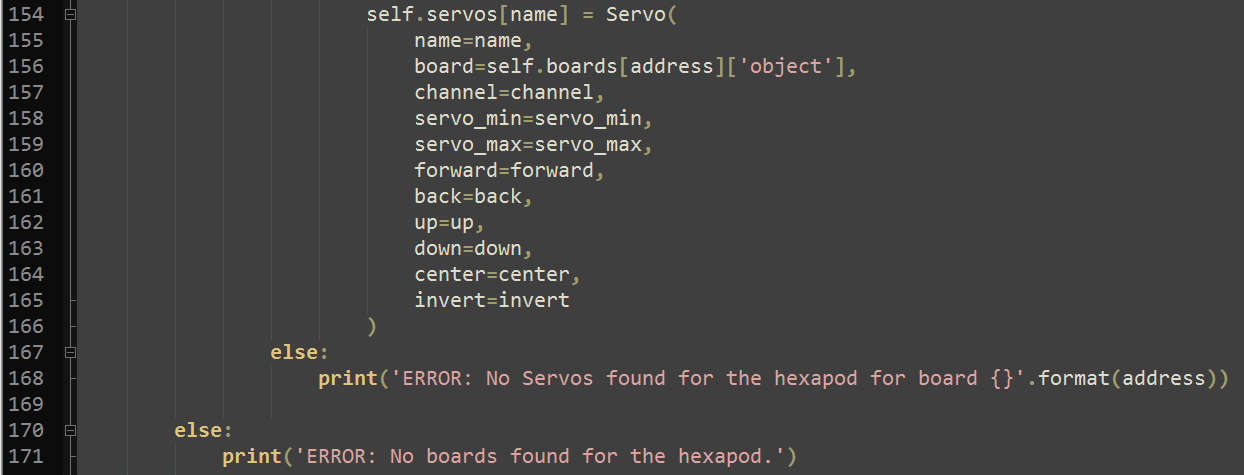


Figure 9: Hexapod constructor (second half)

The Hexapod class contains a lot of methods to encapsulate movement. Several of the methods in the base class are for the lower/upper servos which are 18DOF only. In the future these should be moved to the 18DOF sub-class. The main Hexapod class contains the following methods:

* **resting\_state():** state where legs are close to body, most neutral position for the servos
* **short\_from\_square\_state():** move from square (50% lower, 50% upper) to a short standing position (25% lower, 75% upper), moving uppers first prevents falling
* **short\_from\_resting\_state():** move from the short standing position (25% lower, 75% upper) to the resting position (0% lower, 100% upper), moving lowers first prevents falling
* **square\_from\_tall\_state():** move from the tall standing position (75% lower, 25% upper) to the square standing position (50% lower, 50% upper), moving uppers first prevents falling
* **square\_from\_short\_state():** move from the short standing position (25% lower, 75% upper) to the square position (50% lower, 50% upper), moving the lowers first prevents falling
* **tall\_state():** tall standing position (75% lower, 25% upper), very strenuous on the servos, avoid movement from this state
* **stand():** mapped to the short\_from\_resting\_state() method
* **sit():** mapped to the resting\_state() method
* **rotate():** Utilizes the center legs to rotate the hexapod left/right
* **row():** Utilizes the center legs to move the hexapod forward/backward
* **align\_all\_legs():** Moves all legs to their center position, 90 degrees from the center line of the hexapod
* **center\_all\_legs():** Centers all legs to the 50% position
* **spread\_all\_legs():** Extends all legs to their maximum, front legs to forward position, back legs to their back position, and center legs to their 50%/center position.
* **move\_front\_rotators():** Moves all front rotation servos to the specified position
* **move\_center\_rotators():** Moves all center rotation servos to the specified position
* **move\_back\_rotators():** Moves all back rotation servos to the specified position
* **move\_all\_lowers():** Moves all lower servos to the specified position
* **raise\_all\_lowers():** Moves all lower servos to the up position
* **center\_all\_lowers():** Moves all lower servos to the 50% position, straight out from the hexapod
* **lower\_all\_lowers():** Moves all lower servos to the down position
* **move\_front\_lowers():** Move just the front lower servos to the specified position
* **move\_center\_lowers():** Move just the center lower servos to the specified position
* **move\_back\_lowers():** Move just the back lower servos to the specified position
* **move\_right\_lowers():** Move just the right lower servos to the specified position
* **move\_left\_lowers():** Move just the left lower servos to the specified position
* **move\_right\_left\_right\_lowers():** Move just the right/left/right lower servos to the specified position, used for walking movements
* **move\_left\_right\_left\_lowers():** Move just the left/right/left lower servos to the specified position, used for walking movements
* **move\_all\_uppers():** Move all upper servos to the specified position
* **raise\_all\_uppers():** Move all upper servos to the up position
* **center\_all\_uppers():** Move all upper servos to the 50% position
* **lower\_all\_uppers():** Move all upper servos to the down position
* **move\_front\_uppers():** Move front servos to the specified position
* **move\_center\_uppers():** Move center servos to the specified position
* **move\_back\_uppers():** Move back servos to the specified position
* **move\_right\_uppers():** Move right servos to the specified position
* **move\_left\_uppers():** Move left servos to the specified position
* **move\_left\_right\_left\_uppers():** Move left/right/left servos to the specified position
* **move\_right\_left\_right\_uppers():** Move right/left/right servos to the specified position

The Hexapod\_12DOF sub-class has specific movement methods for the lower number of servos as compared to the 18DOF methods.

* **initial\_tests():** changes the height of the robot from its lowest value, to its tallest value, and back to the lowest
* **sit():** method mapped to lower\_height()
* **stand():** method mapped to center\_height()
* **center\_all\_legs():** empty method to conform to common server interface
* **align\_all\_legs():** empty method to conform to common server interface
* **spread\_all\_legs():** empty method to conform to common server interface
* **lower\_height():** move all legs to the lowest level
* **center\_height():** move all legs to the 50% position
* **raise\_height():** move all legs to the highest position
* **right\_left\_right\_step():** used for walking algorithm
* **right\_left\_right\_step\_back():** used for walking algorithm
* **left\_right\_left\_step():** used for walking algorithm
* **left\_right\_left\_step\_back():** used for walking algorithm
* **turn\_left():** uses right front/back legs to rotate to the left
* **turn\_right():** uses the left front/back legs to rotate to the right
* **row():** uses the center legs to move forward/back
* **rotate():** uses the center legs to rotate the hexapod left/right
* **move\_leg():** helper function to move a legs rotation/raise servos
* **move\_all\_legs():** move all legs to a common rotation/raise value
* **move\_right\_legs():** move right legs to a common rotation/raise value
* **move\_left\_legs():** move left legs to a common rotation/raise value
* **move\_front\_legs():** move front legs to a common rotation/raise value
* **move\_center\_legs():** move center legs to a common rotation/raise value
* **move\_back\_legs():** move back legs to a common rotation/raise value
* **reposition\_front\_legs():** raises, rotates, then lowers front legs
* **reposition\_center\_legs():** raises, rotates, then lowers center legs
* **reposition\_back\_legs():** raises, rotates, then lowers back legs
* **front\_to\_back\_wave():** dancing move to wave front legs front to back
* **side\_to\_side\_wave():** dancing move to wave front legs side to side
* **front\_to\_back\_wave\_back():** dancing move to wave back legs front to back
* **side\_to\_side\_wave\_back():** dancing move to wave back legs side to side
* **front\_leg\_dancing():** moves support to the back and performs front leg dancing moves
* **back\_leg\_dancing():** move support to the front and performs back leg dancing moves

The 18DOF hexapod mimics many of the same methods as the 12DOF methods, but oriented towards the 3 servo per leg design. The key difference and unique methods are listed below:

* **initial\_tests():** moves the hexapod between the resting, short-standing, square-standing, tall-standing positions, and then back down to the resting position, showing the range of motion of the hexapod
* **move\_leg():** moves a legs rotation, lower, uppers servos. Used to abstract the movements of subsequent methods

The design of movements was performed by building up individual movements to more complicated walking/turning/rotating/dancing actions.

#### Server Interface:

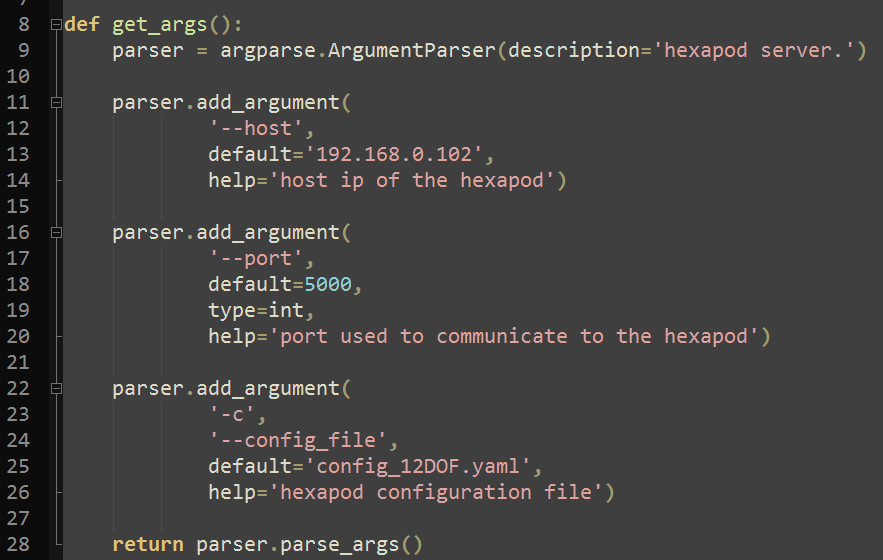


Figure 10: server interface command line argument parsing

The server interface module is located in the github repository in “hexapod/server\_if.py”. The server interface module instantiates a hexapod object and a server socket using a hostname, port, and configuration file. The module uses the Argparse library to manage the command line arguments as seen in figure 10. The module is executed as a script, reading the command line arguments using the get\_args() function and passes the arguments to the Main() function. Main instantiates a hexapod object using the config file (identifying the Hexapod\_12DOF or Hexapod\_18DOF sub-class due to the config file name), instantiates a socket on the host/port specified, and waits for a client to connect. When a client connects to the server the listenToClient() function is called.

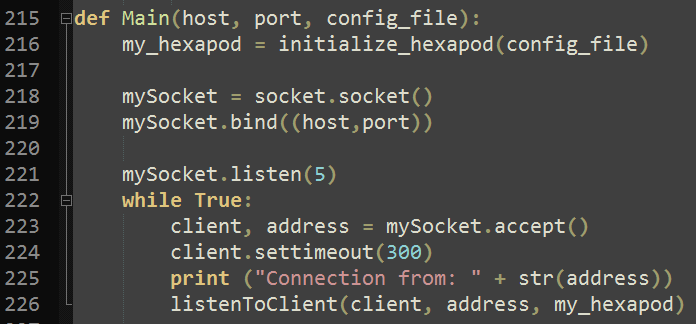


Figure 11: server interface Main() function

listenToClient() reads the incoming message from the client and sends it to the command\_processor() function. The command\_processor() parses the string as a space separated list. If the list contains one item, it is treated as a command with an iteration of 1, if there are two items in the list the first is treated as the command and the second is treated as the iteration count. The commands that are supported are:

* turn\_left/right
* rotate\_left/right
* walk\_forward | walk
* walk\_backward | walk\_back
* front\_dancing\_1/2
* back\_dancing\_1/2
* sit/stand
* center/spread/align

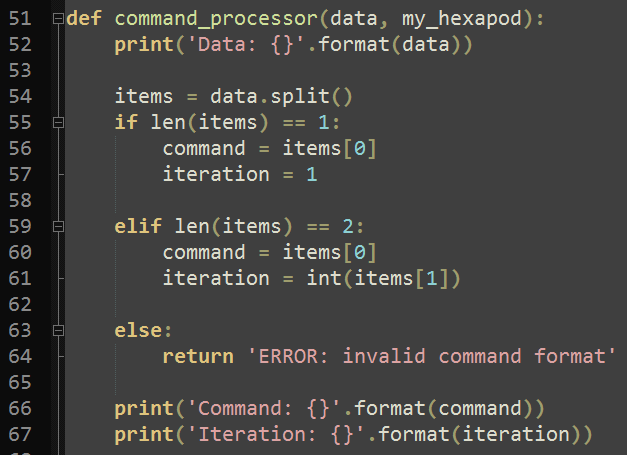


Figure 12: server interface command\_processor() function

Each command takes a number of iterations. Several movements have asymmetrical versions. For these the odd iteration counts are given on version of the movement, and even iterations are given the other version.

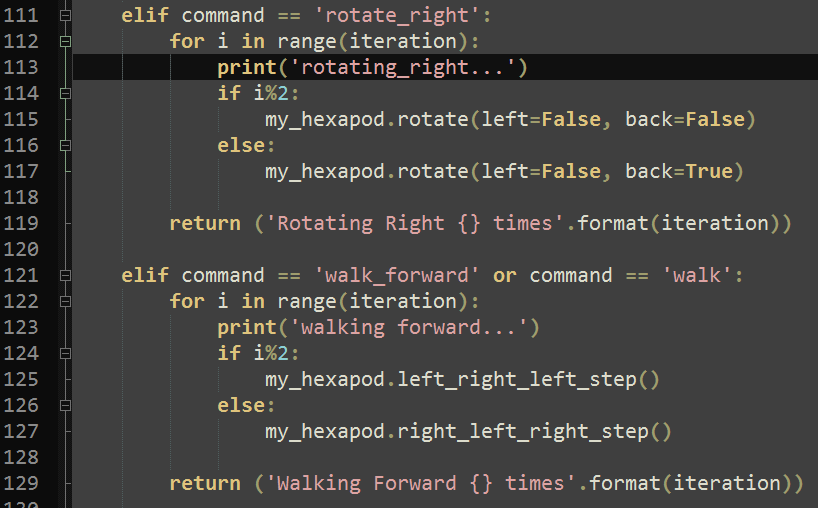


Figure 13: sever interface command\_processor() function demonstrating asymetrical movements

The server interface allows the hexapod to be controlled via any client program that can send it commands. This means that the client could be a user typing each movement, a script processing pre-designed movements, or some other algorithm.

#### Client Interface:

The client interface module is located in the github repository in “hexapod/client\_if.py”. The client code takes several command line arguments controlling how commands are sent to the hexapod via the server interface. These include:

* **host:** host IP address of the machine running the server interface script
* **port:** port used to communicate to the server interface script
* **script:** optional script file to send pre-designed command sequence to the hexapod
* **vision:** optional flag to utilize object\_detection module to control the hexapod

The client interface uses the command line parameters to either: read input from the user until the user enters “q” which quits the program, reads commands from a script file and sends them to the server interface, or calls the object detection modules Main() function with the socket sent as it’s only argument. The object detection module design will be covered in a separate section.

## Stage Manager:

The stage manager design is limited to a Logitech C920 webcam mounted to the top of the stage connected to a laptop running Ubuntu 16.10 with Python3 and OpenCV 3.3 installed. The stage manager host computer runs the client interface code with the “vision” flag running the object\_detection.py module’s Main() function. The Main() function takes input from the webcam, identifies the hexapod and spotlight using HSV color filtering and cleans up the mask using an opening followed by a closing function on each binary image. The center location of the hexapod and spotlight are determined by calculating the moments of the image. <TODO: Add info about moments calculations>

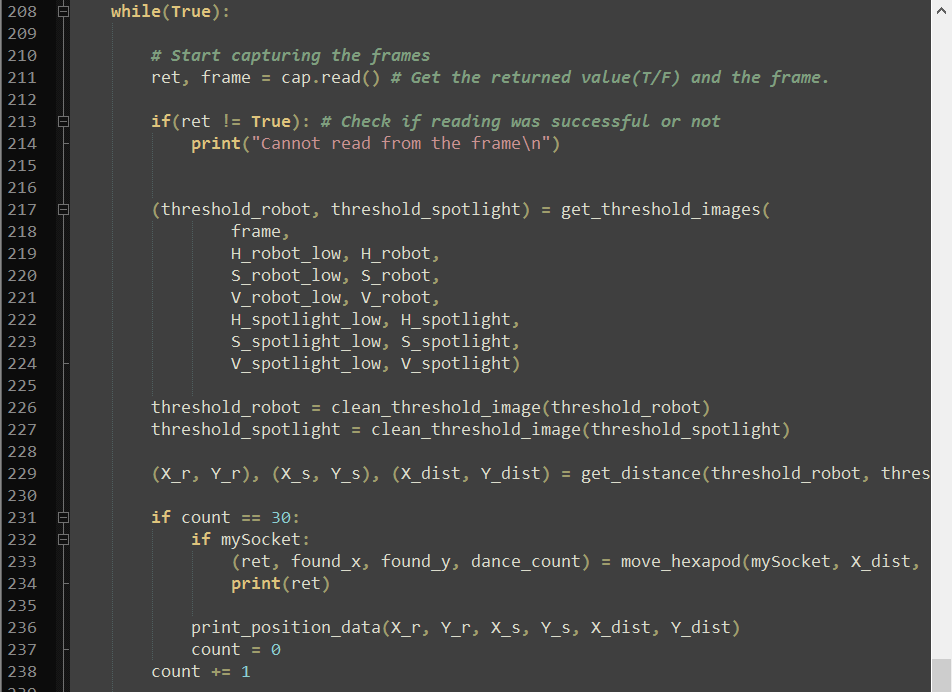


Figure 14: object\_detection.py Main() function while loop

The main while loop of the object\_detection.py code captures a frame from the camera, gets the threshold image based on the robot and spotlights low/high values for HSV values. The clean\_threshold() function performs the opening then closing function to clean-up the threshold image. The get\_distance() function uses the moments method to calculate the X/Y coordinates of the hexapod and spotlight, then calculates the distance between the two in the X-axis and Y-axis. The move\_hexapod() function is called every 30 frames and uses the socket handle, the x and y distance, the found\_x/found\_y Boolean flags and dance count. These are all used to move the hexapod to the spotlight and perform dancing motions under the spotlight.

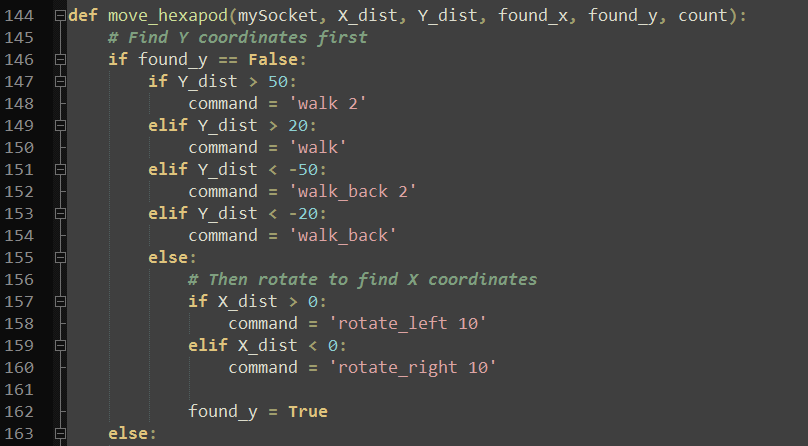


Figure 15: object\_detection.py move\_hexapod() (first half)

The first half of the move\_hexapod function moves the hexapod forward/back until the Y distance is between -20 and 20 pixels from the spotlights Y distance. When the hexapod is within the right Y-distance of the spotlight, it rotates using an iteration count of 10 which was measured to be 90 degrees for the yellow 12DOF hexapod. The Boolean flag “found\_y” is then set to True allowing the second half of the move\_hexapod() function to be executed.

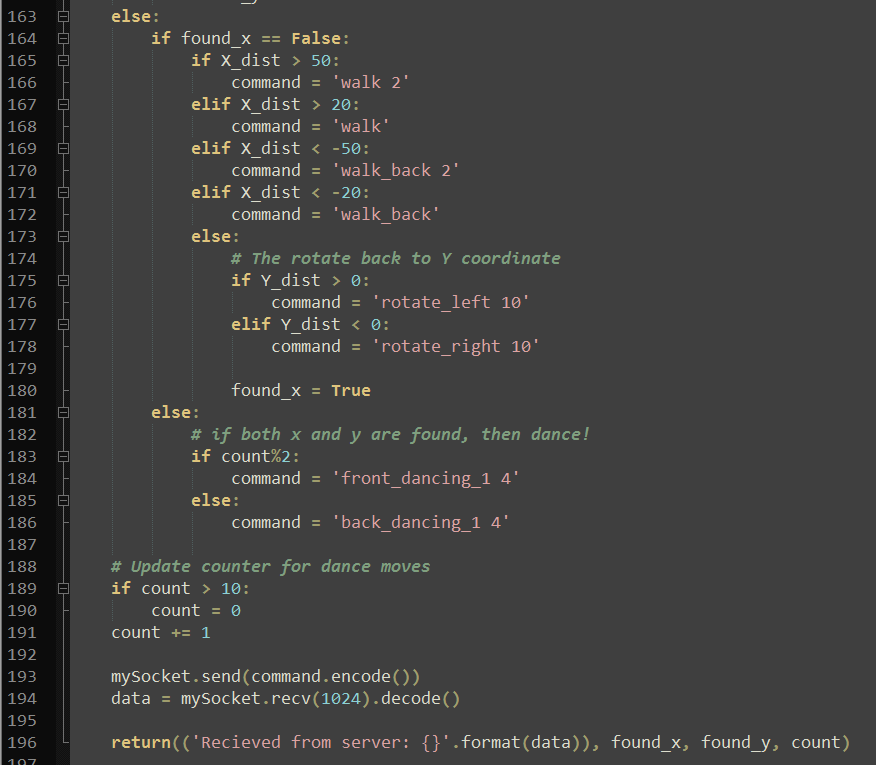


Figure 16: object\_detection.py move\_hexapod() (second\_half)

After the Y distance is found and the robot is rotated to align with the x-axis, the robot moves forward/backward to align the x-axis with the spotlight. When the hexapod is within -20, 20 pixels from the spotlight center the hexapod rotates 90 degrees again, towards the spotlight, then performs a dancing motion. The dancing motion runs using a count that is passed in as an argument and switches from the front\_dancing\_1 motion to the back\_dancing\_1 motion. When any command is determined, the command is sent to the server/hexapod over the socket. In this way the host laptop is able to move the hexapod to the spotlight and dance in the spotlight.

# Results:

The entire project design is available on github here: <https://github.com/vivanbhalla/Dancing-Hexapod>

There are two working hexapods designed (one 12DOF and one 18DOF) that have been tested using the client interface with a sample script. The sample script is located within the github repo in “hexapod/sample\_script.txt”. The 12DOF demonstration is recorded in the video located in “hexapod/Demo/12DOF\_hexapod\_movement\_demo/2017-12-10-12DOF\_hexapod\_demon.mp4”. The 18DOF demonstration is recorded in the video located in “hexapod/Demo/18DOF\_hexapod\_movement\_demo/2017-12-10-18DOF\_hexapod\_demo.m4v”.

The final demonstration was performed with the 12DOF yellow hexapod, and was recorded in two parts, one video of the screencast running the client and server code: “hexapod/Demo/hexapod\_following\_light/hexapod\_demo\_laptop\_screen.m4v”, and one video of the stage while the hexapod moves to the spotlight and dances: “hexapod/Demo/hexapod\_following\_light/hexapod\_demo\_stage\_camera\_compressed.mp4”.

There are several features we would have liked to implement but did not get to…

1. Tracking/moving both hexapods at the same time
2. Moving the spotlights with the pan/tilt controls, allowing the spotlight to track the hexapod
3. More hexapod movements, smoothing the walking/turning motions
4. Add battery power to make robots autonomous
5. Utilize an IMU to increase stability of the hexapods
6. Add other stage components such as speakers and RGB lighting to control the hexapods movements
7. Add hexapod display allowing expressions with the dancing motions
8. Add a camera to the hexapod to allow “robot perspective” as additional feedback/control
9. Develop Kinematics/Inverse Kinematics of robot position/movement algorithms
10. Basic code cleanup and documentation

# Conclusion:

This project has enabled the exploration and development of the hexapod platform in a variety of ways. By developing two unique hexapods the commonality and tradeoffs of each were observed, and the unique programming challenges overcome. These hexapods give a good foundation to further research hexapod movement and control algorithms.

The project also provided the opportunity to investigate computer vision and use camera feedback to control a robot. I see a lot of potential in the “stage as a robot” project and can see expanding the sensors and actuators of the “stage” to include moving spotlights, RGB lighting effects, more robots, and sound effects. What we did do with vision taught us a lot about color space, filtering, and moments.

# References:

Project GitHub: <https://github.com/vivanbhalla/Dancing-Hexapod/tree/master/hexapod/Demo/hexapod_following_light>

SainSmart Hexapod: <https://www.sainsmart.com/products/hexapod-6-leg-spider-robot>

Adafruit 16-channel 12-bit PWM PiHat: <https://www.adafruit.com/product/2327>

Adafruit PCA9685 Python Library: https://github.com/adafruit/Adafruit\_Python\_PCA9685

RaspberryPi: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>

Jupyter Notebook: <https://jupyter-notebook.readthedocs.io/en/stable/>

Python3: <https://docs.python.org/3/>

OpenCV: <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html>

MG996R Servo: <http://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf>

# Appendices:

## 18DOF Hexapod BOM:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Quantity** | **unit cost** | **cost** |
| SainSmart Hexapod Frame | 1 | $ 59.99 | $ 59.99 |
| 10pack of Kuman MG996R servos | 2 | $ 39.99 | $ 79.98 |
| RaspberryPi3 | 1 | $ 34.49 | $ 34.49 |
| Adafruit 16-channel 12-bit PWM controller board | 2 | $ 17.50 | $ 35.00 |
| Total |  |  | $ 209.46 |

## 12DOF Hexapod BOM:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Quantity** | **unit cost** | **cost** |
| RaspberryPi3 | 1 | $ 34.49 | $ 34.49 |
| Adafruit 16-channel 12-bit PWM controller board | 1 | $ 17.50 | $ 17.50 |
| Total |  |  | $ 51.99 |