

ECE 478

**Intelligent Robotics I
Team 2 Final Report**

**Professor:
Marek Perkowski**

**Team #2:
Samuel Salin
Jelon Anderson
Justin Morgan
Surendra Maddula
Saly Hakkoum**

**Date:
December 5th, 2016**

Table of contents:

[Quick Sight on Jimmy Robot](#)

[Our Project](#)

[OpenCV and Image Processing](#)

[Steps to Header and Library File Inclusion for Kinect V1 Visual Studio C++](#)

[A* Pathfinding Algorithm](#)

[UDP: User Datagram Protocol](#)

[Final Integration](#)

[Challenges, problems, and overcomes](#)

[Biography](#)

[Appendix](#)

Chapter 1 : Quick Sight on Jimmy Robot

In the early 21st, Intel Corporation invented a humanoid robot and called it Jimmy. Jimmy is open source and fully customizable. The robot has skeleton, featuring 5052 air plane aluminum brackets, can be reconfigured, and how he looks on the outside is entirely up to the programmer. Jimmy uses 3D printing technology, and it is powered by Intel's Edison chip. Jimmy robot uses an Atom dual core processor and a single core microcontroller, together with WiFi, Bluetooth LE, the required memory, and storage.

This robot is personalized to walk, talk, make hand gestures, use social media channels and more, and according to Jimmy's inventors, "Jimmy helps encourage people to think differently about what a computational device could look like in the future and to spark the imagination of inventors of any age to reimagine the ways in which people can design, create, enjoy and use new digital technologies" (Jimmy' the Humanoid Robot: The New Face of Computing – why section).

In this term, three teams are supposed to be locating and controlling three Jimmy robots by sending the appropriate moving commands to perform a play, and the next chapters will explain in details our part of this play.

Chapter 2 : Our Project

For this term, as members of team #2 we were required to track three Jimmy robots and find the shortest path to move to a specified destination including avoiding obstacles might be in their way.

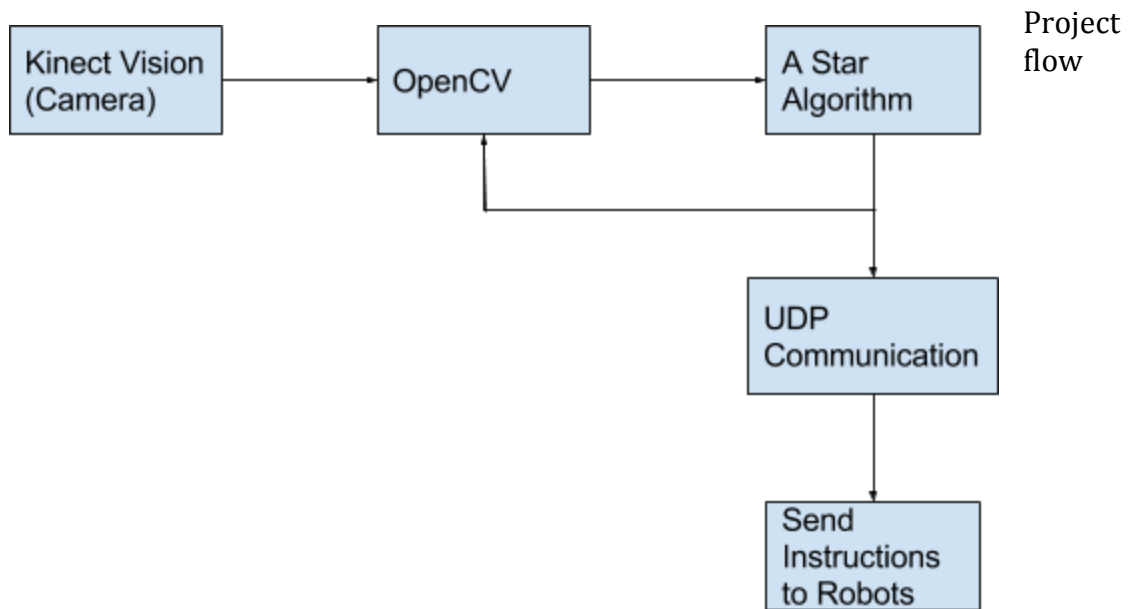
For robots tracking, we used OpenCV library with Kinect. Kinect in our project will capture continues images of our robots locations and movements. OpenCV library will work on these images processing using Tony's code (a code was written by some other students earlier for image processing purposes). See chapter 3 for more details about OpenCV and image processing and chapter 5 for information about the Kinect/Webcam image capturing.

To determine the shortest path for a robot to move to its destination, we used A* algorithm which is the process of plotting an efficiently traversable path between multiple points, called nodes. This algorithm will move the robots away from obstacles if they are found in the way to the destination (See chapter 4 for more details.)

In order to send commands to the robots to move through the shortest path found by A* algorithm, a user datagram protocol (UDP) is used. UDP is the simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves

minimum amount of communication mechanism. Commands will be set to each robot through this protocol to move to their destination locations. (See chapter 6 for more details.)

All of the parts above are integrated to work nicely together in the final stage to demonstrate a completion of our project for fall term 2016.



Chapter 3 : OpenCV and Image Processing

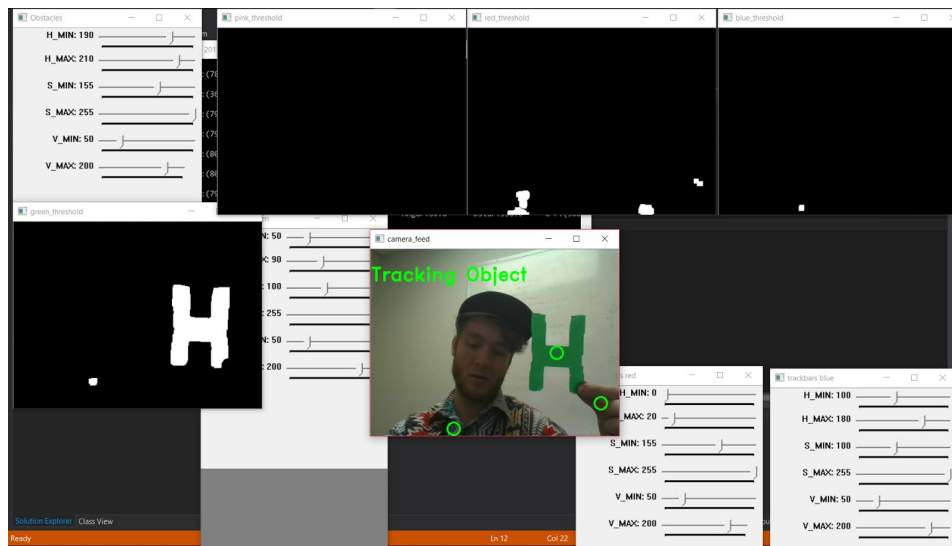
Since our project required image processing to determine the location of robots and obstacles and there is a preponderance of tutorials for openCV on the internet we decided to use it as the basis for our image processing. OpenCV stands for open computer vision, and it is an open source library that is very easy to work with in Python and C++. We used C++ because at the beginning of the term we were fortunate enough to find a code base, written by Tony Muilenburg, that already could recognize objects of four different colors which was written in C++. We decided that the three jimmy robots would have different colored hats in order to differentiate between them and make it easy to figure out their respective locations.

Since there were Three jimmy robot and four color's already able to be detected it was decided that the fourth color, pink, would be used to denote obstacles. OpenCV makes it easy enough to draw bounding circles and rectangles around objects, which in turn are readily passed to the pathfinder as either a center and radius or top left corner and bottom right corner to allow the pathfinder to determine the spaces where an obstacle is present.

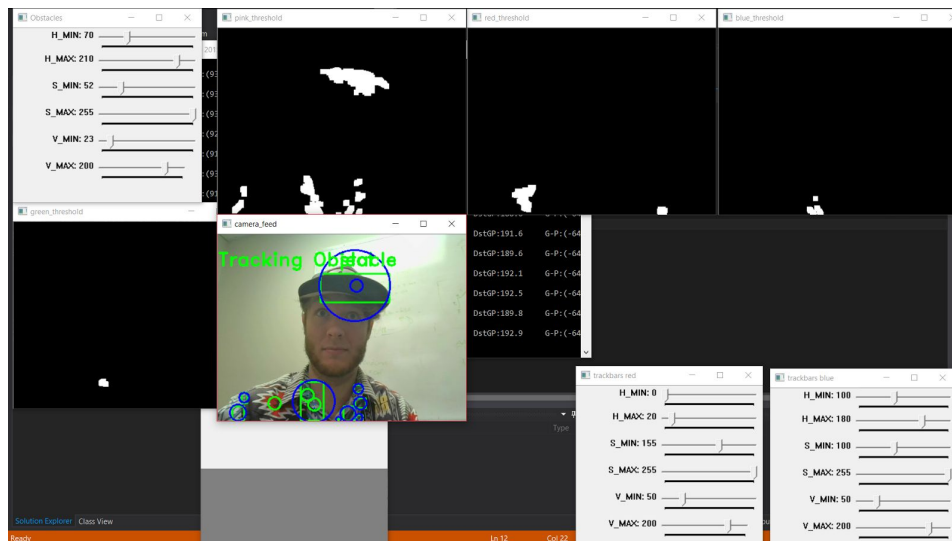
The way the program works is very straightforward. First OpenCV opens a camera to use as a video capture source. it then dilates and erodes the source to convert it into something simpler and therefore easier to process. Simply blurring the video feed was also tried, but it was too resource intensive. once the image has been prepared, it is sent to a series of callback functions that use Hue, Saturation, and Vibrance (HSV) values to filter the image into specific colors. Then, for the colors corresponding to robots, some calculations are performed to determine where the middle coordinates of said object are located. these calculations were already being done in Tony's code and we were not able to understand the way they worked.

Red, Blue, and Green all use the same callback function called `track_object` since they all need to do the same processing. Obstacles use a different function called `track_obstacle` which uses contour-based image processing to draw the outline of objects and then find the minimum bounding rectangle and circle.

One cool feature of the image processing is that it can handle any number of obstacles. it could probably handle moving obstacles as well but that theory was never tested and it would require frequent use of the pathfinder, which would slow everything down immensely. theoretically it could handle any number of distinct red, blue, and green robots, but controlling them individually and keeping track of which one is which would be difficult.



OpenCV
recognizing
and finding
the middle
of a green
object



OpenCV
recognizing
some
obstacles

Chapter 4 : Steps to Header and Library File Inclusion for Kinect V1 Visual Studio C++

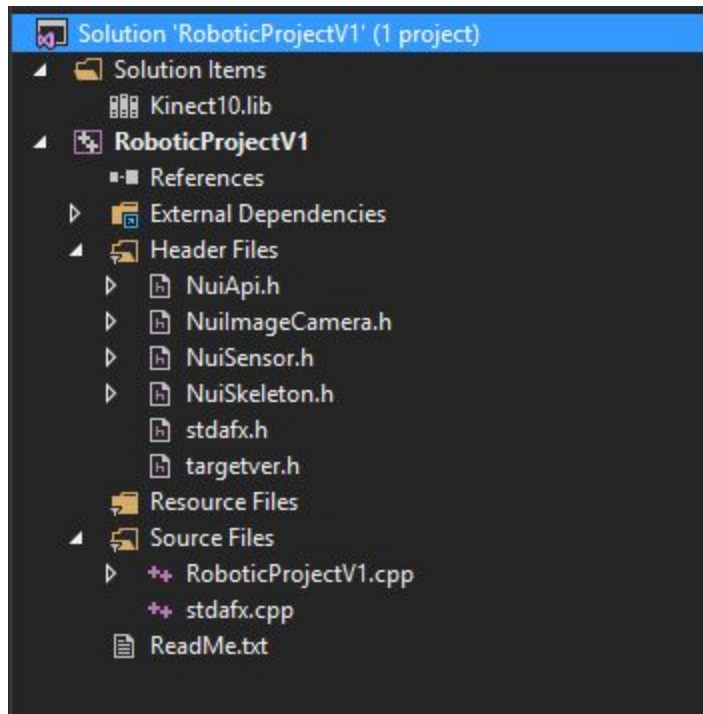
The below steps were formed by working together with other team's people who also needed kinect.

Follow the following steps to avoid any error w.r.t Nui header and library files for Kinect V1 To run Kinect successfully using Visual Studios:

First steps would be installing the softwares that are need to detect the kinect connected to your PC. Latest Kinect studio software and the Developer Toolkit browser software which is compatible with the earlier installed Kinect needs to be installed first, and then your kinect is ready.

We need to install latest Kinect studio and the Most important thing when you attach Kinect (NUI) header files like NuiApi.h or NuiImage Camera.h you should make sure the following things are added to you project.

1. Header Files necessary. These can be found in the following default directory: C:\Program Files\Microsoft SDKs\Kinect\v1.8\inc
2. Addition of Kinect library to you solution items. For this just right click on your project and add the library file directly. This file can be found in C:\Program Files\Microsoft SDKs\Kinect\v1.8\lib\x86
3. Adding Include and Library directories. For this follow the following steps:
 - Right click on you project in the solutions explorer window found on the right hand side of your screen In configuration properties VC++ directories
 - Add this C:\Program Files\Microsoft SDKs\Kinect\v1.8\inc directory in the include directories and Add this C:\Program Files\Microsoft SDKs\Kinect\v1.8\lib\x86 directory to the Library directories.
4. Now in go to Linker in Configuration properties à General à In additional Library dependencies add this C:\Program Files\Microsoft SDKs\Kinect\v1.8\lib\x86
5. Now in go to Linker in Configuration properties à Input à In additional Library dependencies add this C:\Program Files\Microsoft SDKs\Kinect\v1.8\lib\x86\Kinect10.lib
6. Finally your Solution Explorer should look like this:



Chapter 5 : A* Pathfinding Algorithm

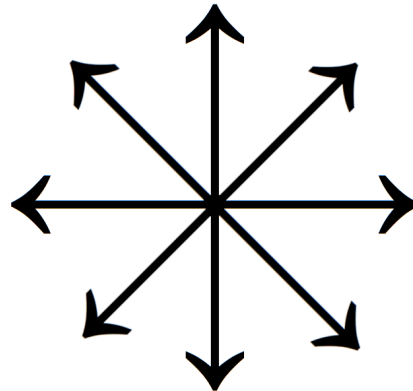
Upon receiving data from the kinect sensor, our jimmy robots will need to have a path generated for them. One common approach, used in many video games for AI, is the A-Star pathfinding algorithm. A-star A* is a best first search algorithm, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that *appear* to lead most quickly to the solution. It measure between two points called nodes. The code itself uses a series of priority queues to search each node. The goal is to find the best path by searching the entire area. Sometimes, this can be slow.

Initially the A star algorithm was implemented in C++ code with the intention of querying openCV code. However, for reasons unknown the time it took to complete was immense, sometimes 30 seconds for 100x100 grid search. Jelon found a version that was functionally (and in some ways appeared) an identical translation of C++ code to Python code. This python code ran approximately 38 times faster. The decision then was made to switch over to running python code. This introduces a multitude of new challenges.

There are several steps needed for the code to run with a python interpreter.

1. Visual studio 2015 Community
2. Mode must be in RELEASE x64 (openCV only works in 64 bit)
3. "python.h" file must be included at top of source file
4. solution properties must have Python27/include as include directories
5. Solution properties must have Python27/libs as Linker library directories
6. Python27 install MUST be 64 bit only, or else your linker will not work properly.

The pathfinder itself is very robust. No matter the shape of obstacles, or the amount, as long as both the start and finish are not occupied by an obstacle, and there is no solid wall between the two terminuses, the pathfinder will create a route. How the program works in regard to this path is that it requests a string, containing the route. The string with the route data is a series of integers ranging from 0 to 7, for the 8 possible directions our robot can travel in. These directions are always relative to the layout of the arena, not the robot. The figure at right



demonstrates this behavior. The route may look like “4444333355554444” ect. The code will then parse this direction list and mark the route on the map. The second part of this route is handled in integration, for the UDP commands. A star algorithm itself uses a set of nodes, much like graphs, and analyzes the movement cost between two points. It uses priority queues to see what the cost of the current and previous node is. The algorithm functions as

$$Cost_of_edge + Cost_of_previous_node + Estimated_cost_to_reach_target_from(node)$$

This will push the search to find the path with fewest visited nodes.

The picture below is a snapshot of what the pathfinder looks like internally. Usually, mapping is disabled since it really slows down our performance. But for testing and verification, it is extremely handy to have.

[illegible]

You can see even with multiple obstacles clouding the field, the robot will be able to find a way between them. The A star algorithm can be negatively affected by larger map resolution, since it treats space on the field as an index that must be searched. anything above 200x200 will probably be too slow to function in this case. In the source code this python pathfinder function also does the mapping as well. While one can map in c++, it is redundant. You can also suppress this output by simply commenting out the mapping print statements, but it is useful to have to see how the algorithm is adjusting. Since the intention is to have this repeatedly drawn, the route would get shorter and shorter until the jimmy robot is at it's destination. this also implies that moving obstacles are supported based on the programs need to redraw the arena when pathfinding.

Chapter 6 : UDP: User Datagram Protocol

User Datagram Protocol, UDP is a communication protocol that operates at a lower latency and loss tolerating connection between applications and the internet. UDP is an alternative communication method to Transmission Control Protocol, or TCP. Both UDP and TCP are internet protocols used to send short packet of data. These packets is also referred as datagrams. The major disadvantage of UDP is that if data are lost or didn't get through during the communication, the receiver may not respond to the communicated data. Moreover, data packets may also be received out of order. This is a major disadvantage because delayed data can cause confusion to the client. In the spectrum of robots, a delay message can create unpredictable behavior during the course of the robot theater.

In the scope of our project. UDP data will be used to transmit short instruction packets from the pathfinding algorithm to each Jimmy robot. The command format for the data packet is <robot_color> <instruction command>. And example would be "red Move Forward". Furthermore, it's our goal to create a system that will send data to multiple robots simultaneously to create a fluid jimmy theater or any other jimmy interactions.

To establish a valid communication protocols, it needs a client and server script. The server script behaves like the command center that is used to send data to the client. Since each robot has its own Raspberry Pi, so it has its own unique IP address. Because of this, each Raspberry pi (or robot) is a client. For the server script, this single script communicates to all the clients. Within this server script, it chooses which robot to establish a connection by binding with the IP address and a port number. The port number can be any port that is greater than 1000, or non-reserved port. Data will be sent via this port to the specified IP address. Since the client scripts are constantly running on the jimmy. In other words, their connection is always open and waiting for data packets. The server script just toggles between binding connections with each client and send out the individual data packets.

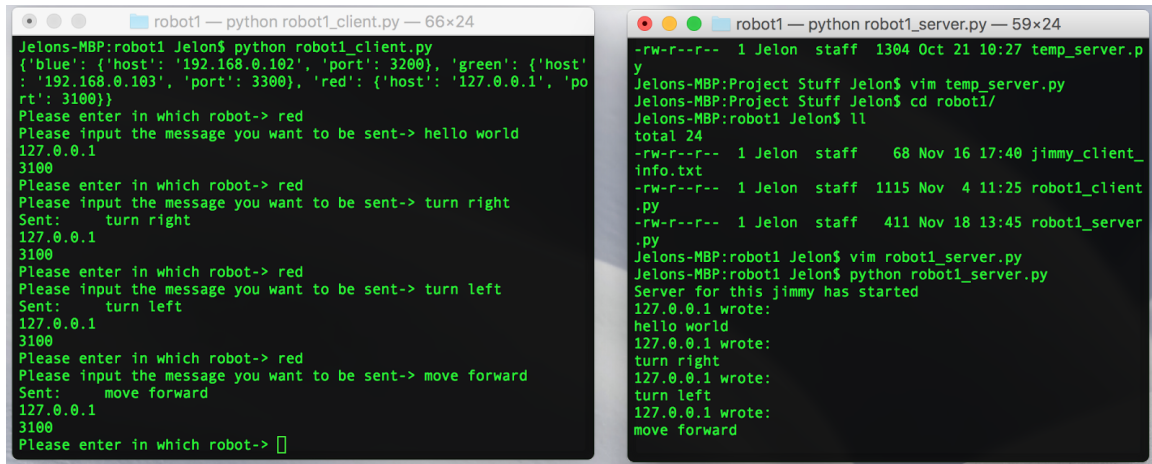
```

graph LR
    Server[UDP Server (Send Data)] --> Client1[UDP Client (Jimmy 1)]
    Server --> Client2[UDP Client (Jimmy 2)]
    Server --> Client3[UDP Client (Jimmy 3)]
  
```

The diagram illustrates a one-to-many communication pattern using UDP. A single box on the left, labeled "UDP Server (Send Data)", has three arrows pointing to three separate boxes on the right. These boxes are labeled "UDP Client (Jimmy 1)", "UDP Client (Jimmy 2)", and "UDP Client (Jimmy 3)". This represents a server broadcasting data to multiple clients simultaneously.

[illegible]

Here are some sample outputs for one of the robots. The left window shows server script selecting which robot and the data to be sent. The right window shows the client receiving the data packet.



```
robot1 — python robot1_client.py — 66x24
Jelons-MBP:robot1 Jelons-MBP$ python robot1_client.py
{'blue': {'host': '192.168.0.102', 'port': 3200}, 'green': {'host': '192.168.0.103', 'port': 3300}, 'red': {'host': '127.0.0.1', 'port': 3100}}
Please enter in which robot-> red
Please input the message you want to be sent-> hello world
127.0.0.1
3100
Please enter in which robot-> red
Please input the message you want to be sent-> turn right
Sent: turn right
127.0.0.1
3100
Please enter in which robot-> red
Please input the message you want to be sent-> turn left
Sent: turn left
127.0.0.1
3100
Please enter in which robot-> red
Please input the message you want to be sent-> move forward
Sent: move forward
127.0.0.1
3100
Please enter in which robot->

robot1 — python robot1_server.py — 59x24
-rw-r--r-- 1 Jelons staff 1304 Oct 21 10:27 temp_server.py
Jelons-MBP:Project Stuff Jelons-MBP$ vim temp_server.py
Jelons-MBP:Project Stuff Jelons-MBP$ cd robot1/
Jelons-MBP:robot1 Jelons-MBP$ ll
total 24
-rw-r--r-- 1 Jelons staff 68 Nov 16 17:40 jimmy_client_info.txt
-rw-r--r-- 1 Jelons staff 1115 Nov 4 11:25 robot1_client.py
-rw-r--r-- 1 Jelons staff 411 Nov 18 13:45 robot1_server.py
Jelons-MBP:robot1 Jelons-MBP$ vim robot1_server.py
Jelons-MBP:robot1 Jelons-MBP$ python robot1_server.py
Server for this jimmy has started
127.0.0.1 wrote:
hello world
127.0.0.1 wrote:
turn right
127.0.0.1 wrote:
turn left
127.0.0.1 wrote:
move forward
```

Both server and client scripts are written in python. The specific version python used is 2.7. To my knowledge, Python 3.0 or later does not support my current code due to changes of libraries between versions, and some libraries may not even be supported or other issues including syntaxes. The command used to run these python scripts on command window is `python <file_name>`. To run script in IDE, F5 is the command to run python scripts. To pass arguments to python script using command line, arguments are listed after `python <file_name> arg1 arg2` etc. Current, I do not know how to pass arguments using IDE on Windows.

Chapter 7 : Final Integration

There are many parts to integration, and it is unfortunately an ongoing process due to the multiple components. The flow goes as this. OpenCV code contains the data we see, and passes it to the pathfinder in python. Python's pathfinder will return a string containing our route. To perform this we must set up the interpreter. In C++, we fire up openCV, while the track bars have been created and the camera is monitoring the field, it will stay in a infinite while(1) loop. At the end of this loop, we will run our interpreter. It can locate the robot(s), any obstacles it sees based on the color, and add bounding circles. These obstacles are packaged in a format that can be sent to python.

Unfortunately at the time of this writing that format has not been discovered, due to unforeseen difficulties in utilizing Python C API. By sending the obstacle data, python can draw the map based on what openCV can see, and produce a path. The returned value of this path is simply a long string of numbers.

We iterate through that string. We send the direction (could be 4, 5, 6 ect.) which is an int that is sent to a function in python called CreateCommandTarget. Command target works by maintaining a 2 index list of directions, where Com_del = [current direction, previous direction]. Thus, we need another instance of the python interpreter. The delegator will function by seeing how these two directions compare. If, for example, the direction for current and old is 3, it means the jimmy is still walking in a straight line. We do not alters its course. If the current direction is greater than the previous direction by 1, or 2, say we have 5 where we were previously moving in direction 4, we turn right either 45 degrees or 90 degrees. Conversely, if current direction is less than previous direction by 1 or 2, we will turn left. These cases are analyzed by a large elif block, pictured here. This is the portion of the command delegator that is critical to proper jimmy motion.

The set of commands is sent as a string in this case (for debug, as a real command it would be in a different format) and is sent off to the Client send function which establishes a connection with one of the robots and sends the desired set of commands. much experimentation is needed to determine how long the jimmy walks, how fast, the resolution of the arena, and so on. Once the command is sent, the interpreter returns back to C++ code to restart the loop and begin the process over once more. The camera should see the jimmy's new location, and of course the obstacles if possible. It will then resend these. The only variable constant in all of this is the goal coordinates. Gradually, the route will become shorter as many passes are made. While this solution is not yet optimized for speed, it set up means that it can be reasonably implemented with few extra work. When the start is the same as the finish coordinates it will terminate.

```

def Client_send(msm_string):
    host = '127.0.0.1'
    port = 3100
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto(msm_string + "\n", (host, port))
    s.close()
def CreateCommandTarget(Dir,Com_Del,Com_List):
    global count
    msm = ""
    if not Com_Del:
        Com_Del.append(Dir)                #list is empty, add to list [cur, prev]
        return
    elif len(Com_Del) == 1:
        Com_Del.insert(0,Dir)
    else:
        Com_Del.insert(0,Dir)
        Com_Del.pop(2)
    print Com_Del
    c = Com_Del[0]
    p = Com_Del[1]
    if ( c == p ):
        count += 1
        msm = "1"
        print "MoveForward"
    elif ( p - c == 1 and c != 0 ):
        count += 1
        print "Turn left 45, Move forward"
        msm = "2 45"
    elif ( p - c == 2 and c != 0 ):
        count += 1
        print "Turn left 90, Move forward"
        msm = "2 90"
    elif ( p - c == 3 and c != 0 ):
        count += 1
        print "Turn left 135, Move forward"
        msm = "2 135"
    elif ( c - p == 1 and c != 0 ):
        count += 1
        print "Turn right 45, Move forward"
        msm = "2 -45"
    elif ( c - p == 2 and c != 0 ):
        count += 1
        print "Turn right 90, Move forward"
        msm = "2 -90"
    elif ( c - p == 3 and c != 0 ):
        count += 1
        print "Turn right 135, Move forward"
        msm = "2 -135"
    elif ( c == 0 and p == 7) or (p == 0 and c == 1):
        count += 1
        print "Turn right 45, Move forward"
        msm = "2 -45"
    elif ( c == 0 and p == 6) or (p == 0 and c == 2):
        count += 1
        print "Turn right 90, Move forward"
        msm = "2 -90"
    elif ( c == 0 and p == 5) or (p == 0 and c == 3):
        count += 1
        print "Turn right 135, Move forward"

```

```

        msm = "2 -135"
    elif ( c == 0 and p == 1) or (p == 0 and c == 7):
        count += 1
        print "Turn left 45, Move forward"
        msm = "2 45"
    elif ( c == 0 and p == 2) or (p == 0 and c == 6):
        count += 1
        print "Turn left 90, Move forward"
        msm = "2 90"
    elif ( c == 0 and p == 3) or (p == 0 and c == 5):
        count += 1
        print "Turn left 135, Move forward"
        msm = "2 135"
    print msm
    Client_send(msm)                                     #send commands to UDP delegation

#def draw_map(the_map,m,n):

```

Path-finding and future integration plans:

As it stands, the current iteration of software is in pieces rather than one whole unit. But with this roadmap, integration can be achieved relatively easily. The first step has already been mostly completed. OpenCV will use the interpreter to call python and compute a path for the jimmy. The returned string will then be parsed one by one, in our case we will take the first index and provide information for a second python call. Using the interpreter, we have two choices. The easy but clunky solution is sending two integers; one of the current direction, and the other is the previous direction if applicable. This is because we used a two-indexed list in python to compare. If our pair of directions suits one of up to 13 (maybe more) cases, we send a series of commands to the jimmy. If one were to figure out Numpy arrays in python C API, you can simply send and return lists. This will all work for one robot. To add for additional robots, extra data will be needed, mainly the coordinate locations of starting and goal positions. The Command delegator will need additional variable to indicate the robot the command is being sent to, and which list to compare (of current and previous directions) . OpenCV's program flow must be modified as well. Currently, everything is hard coded in for testing purposes. The start, the end, the size of obstacles, ect. All will now need to be created from what the camera sees and what OpenCV creates. The obstacles will ideally be sent in the form of a multidimensional array of Coordinates, x,y of center, and radius of circle. Ideally, placing all this python code at the end of the massive while (1) loop in OpenCV, having it parse, will then repeat. One will have to calibrate the time it takes for this operation to complete and the time it takes for the jimmies to move the desired amount, to see how often the pathfinder should run.

Chapter 8 : Challenges, problems, and overcomes

One challenge we had was communication between robots. The current state of UDP is that there is a single server that talks to the three jimmy clients. One of the suggested goals from Melih is to develop a system where the robots can issue instructions to each other. I tried creating server and client script for each robot and have all the scripts running simultaneously. However, binding was a continuous problem and was never resolved.

Making OpenCV draw bounding shapes turned out to be a huge challenge and took weeks. the image processing library has a series of contour-related functions and it turned out one needs to draw the contours before trying to find bounding shapes.

Making C++ and python play nice together turned out to be a giant hassle. It is encouraged to read up on Python C API and investigate into Numpy, which is a 3rd party utility one would have to install. Numpy has many functions in its libraries that concern converting c++ style arrays and pointers to usable python objects. Being able to successfully pass arrays to python is a huge step forward, since many types of data can be expressed in multidimensional arrays.

One must also add the python interpreter for command delegation, by parsing the route string we have received from pathfinder and sending directional integers to be translated. This part is more trivial, since the function call for python only contains integers.

One of the challenges we faced in the integration stage of this project was passing arguments and parameters between C++ and Python codes as mentioned before. One way we could have used and highly suggest for people developing this project to overcome this problem, is to use ROS (Robot Operating System). ROS is a collection of software used for software development. The main client libraries for ROS are Python, C++, LISP, However, it also supports other programming languages. This operating system could have saved us time and effort in the final stage of completing our project, but unfortunately it was discovered too late in the term to be used.

Chapter 9 : Bibliography

- A* Search Algorithm: https://en.wikipedia.org/wiki/A*_search_algorithm
- A * algorithm Python implementation code source
<http://code.activestate.com/recipes/577519-a-star-shortest-path-algorithm/>
- About Jimmy: <http://www.21stcenturyrobot.com/about-jimmy>
- ‘Jimmy’ the Humanoid Robot: The New Face of Computing:
http://download.intel.com/newsroom/kits/makers/pdfs/Jimmy-robot_factsheet.pdf
- UDP (User Datagram Protocol): <http://searchsoa.techtarget.com/definition/UDP>
- User Datagram Protocol: https://en.wikipedia.org/wiki/User_Datagram_Protocol
- http://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html
- http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html

Appendix

Project github link: <https://github.com/Salyhakkoum/ECE-478>

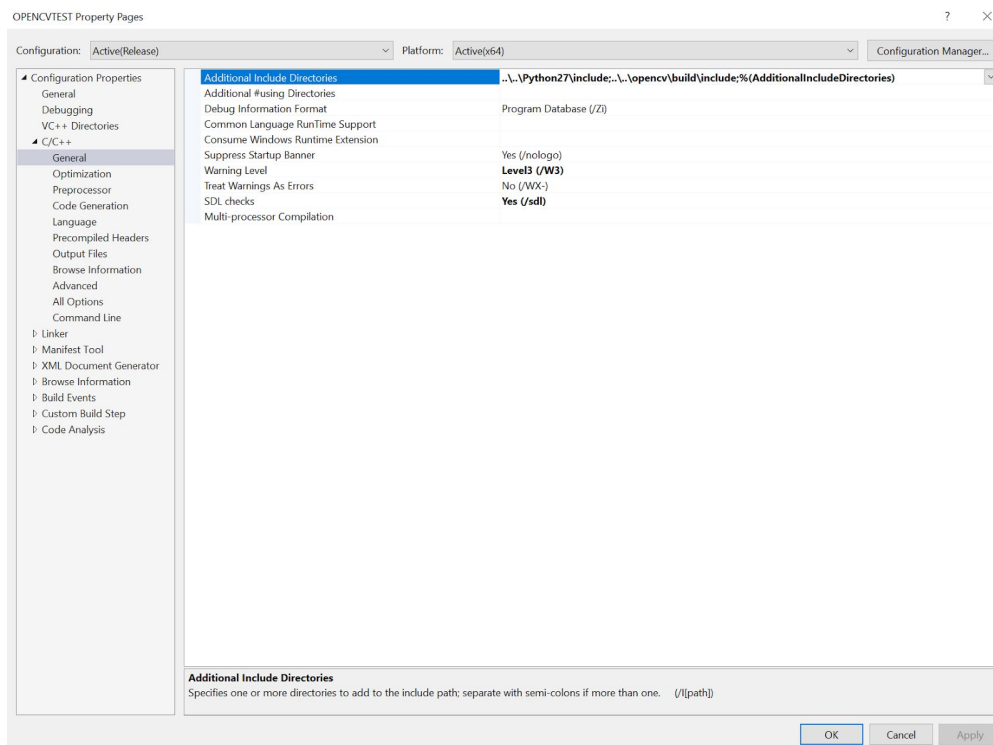
Packages that need to be installed

64 bit python

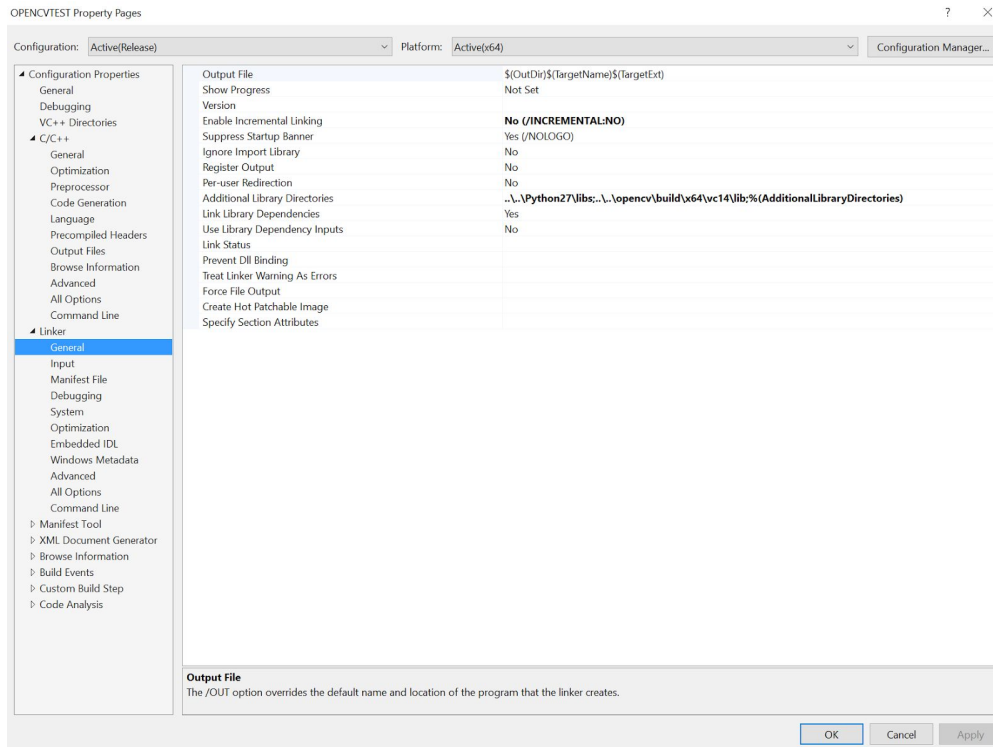
OpenCV version 3.1

Visual studio (not strictly needed)

if the github project is downloaded and then the OPENCVTEST visual studio project is opened everything should work. For general include directories have Python27 (64bit installed) as an include directory, in addition of openCV\build\include.



For the linker libraries, one need Python27\libs added, as well as opencv\build\x64\vc14\lib for the linker. You must have a python 64 bit library, or else the linker will fail to compile. We have installed a 64 bit install here inside the repository but you can point these linker libraries to your desired install.



Another common issue with running OpenCV code in this project is the error message where, despite successful compilation, running the code with debugging will trigger a opencv_world301d.dll file not found. In our case here, we directly inserted this file into the output directory of our project. this is where the .exe file for the code is created. Not that all the information pertaining to Python installs requires that one must install the 64-bit python somewhere on their PC, anywhere is sufficient. The python included in the repository is for simplifying the setup process by providing include paths. However, having this alone does not install python on the windows system PATH (which runs these tools individually) which is why user must install on their own. To access python interpreter code for development, checkout the Pytohn_interpreter_EXPERIMENTAL branch on the repository.

UDP Code:

jimmy_client_info.txt

```
red,192.168.0.101,3100
blue,192.168.0.102,3200
green,192.168.0.103,3300
```

udp_client.py

```
import socket
import sys

def Main():
    print type(sys.argv[1])          // system input
    robot_info = {}                  // Dictionary hold robot info
    # Open jimmy info file and create dictionary
    with open("jimmy_client_info.txt","r") as fin:
        for line in fin:
            line = line.strip('\n')
            parts = line.split(',')
            temp = {}
            temp['host'] = parts[1]
            temp['port'] = int(parts[2])
            robot_info[parts[0]] = temp

    print robot_info

    #robot = raw_input("Please enter in which robot-> ")    // Prompt for input
    #data = raw_input("Please input the message you want to be sent-> ")
    robot = sys.argv[1]          // parse for system inputs
    data = sys.argv[2]

    while robot != "q":          // Valid robot color input
        try:
            host = robot_info[robot]['host']                // Parse robot IP and port
            HOST = '{}'.format(host)
            PORT = robot_info[robot]['port']
            print HOST
            print PORT

            // Create UDP connection and send message via port to IP
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.sendto(data + "\n", (HOST, PORT))
            robot = raw_input("Please enter in which robot-> ")
            data = raw_input("Please input the message you want to be sent-> ")
            print "Sent: {}".format(data)
        except:
            // Invalid robot input.
            print "Please input a value robot name"
            robot = raw_input("Please enter in which robot-> ")
            data = raw_input("Please input the message you want to be sent-> ")

    if __name__ == "__main__":
```

```
Main()
```

robot1.py

```
import SocketServer

class MyUDPHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        data = self.request[0].strip()           // Receive data from server
        socket = self.request[1]
        print "{} wrote:".format(self.client_address[0])
        print data

if __name__ == "__main__":
    print "Server for this jimmy has started"
    HOST, PORT = "192.168.0.101", 3100           // Establish IP and port for Jimmy
    server = SocketServer.UDPServer((HOST, PORT), MyUDPHandler) //Bind connection
    server.serve_forever()
```

robot2.py

```
import SocketServer

class MyUDPHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        data = self.request[0].strip()           // Receive data from server
        socket = self.request[1]
        print "{} wrote:".format(self.client_address[0])
        print data

if __name__ == "__main__":
    print "Server for this jimmy has started"
    HOST, PORT = "192.168.0.102", 3200           // Establish IP and port for Jimmy
    server = SocketServer.UDPServer((HOST, PORT), MyUDPHandler) //Bind connection
    server.serve_forever()
```

robot3.py

```
import SocketServer

class MyUDPHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        data = self.request[0].strip()           // Receive data from server
        socket = self.request[1]
        print "{} wrote:".format(self.client_address[0])
        print data

if __name__ == "__main__":
    print "Server for this jimmy has started"
    HOST, PORT = "192.168.0.103", 3300           // Establish IP and port for Jimmy
```

```
server = SocketServer.UDPServer((HOST, PORT), MyUDPHandler) //Bind connection
server.serve_forever()
```

OpenCV Code:

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
// #include <opencv2\opencv.hpp>
#include <math.h>

using namespace std;
using namespace cv;
int camera_number = 0;
int object_found = 0;

//threshold values for hsv
int H_MIN_red = 0;
int H_MAX_red = 20;
int S_MIN_red = 155;
int S_MAX_red = 255;
int V_MIN_red = 50;
int V_MAX_red = 200;

int H_MIN_blue = 100;
int H_MAX_blue = 180;
int S_MIN_blue = 100;
int S_MAX_blue = 255;
int V_MIN_blue = 50;
int V_MAX_blue = 200;

int H_MIN_green = 50;
int H_MAX_green = 90;
int S_MIN_green = 100;
int S_MAX_green = 255;
int V_MIN_green = 50;
int V_MAX_green = 200;

int H_MIN_pink = 190;
int H_MAX_pink = 210;
int S_MIN_pink = 155;
int S_MAX_pink = 255;
int V_MIN_pink = 50;
int V_MAX_pink = 200;

void my_trackbars(int, void*) { // called when trackbars are instantiated
}

void createTrackbars() {
    //create window for trackbars
    namedWindow("trackbars red", WINDOW_NORMAL);
    resizeWindow("trackbars red", 300, 300);
    //create memory to store trackbar name on window
    char TrackbarNamer[50];

    //create trackbars and insert them into window
    createTrackbar("H_MIN", "trackbars red", &H_MIN_red, 255, my_trackbars);
```

```

createTrackbar("H_MAX", "trackbars red", &H_MAX_red, 255, my_trackbars);
createTrackbar("S_MIN", "trackbars red", &S_MIN_red, 255, my_trackbars);
createTrackbar("S_MAX", "trackbars red", &S_MAX_red, 255, my_trackbars);
createTrackbar("V_MIN", "trackbars red", &V_MIN_red, 255, my_trackbars);
createTrackbar("V_MAX", "trackbars red", &V_MAX_red, 255, my_trackbars);

//create window for trackbars
namedWindow("trackbars blue", WINDOW_AUTOSIZE);
//create memory to store trackbar name on window
char TrackbarNameblue[50];

//create trackbars and insert them into window
createTrackbar("H_MIN", "trackbars blue", &H_MIN_blue, 255, my_trackbars);
createTrackbar("H_MAX", "trackbars blue", &H_MAX_blue, 255, my_trackbars);
createTrackbar("S_MIN", "trackbars blue", &S_MIN_blue, 255, my_trackbars);
createTrackbar("S_MAX", "trackbars blue", &S_MAX_blue, 255, my_trackbars);
createTrackbar("V_MIN", "trackbars blue", &V_MIN_blue, 255, my_trackbars);
createTrackbar("V_MAX", "trackbars blue", &V_MAX_blue, 255, my_trackbars);

//create window for trackbars
namedWindow("trackbars green", WINDOW_AUTOSIZE);
//create memory to store trackbar name on window
char TrackbarNamegreen[50];

//create trackbars and insert them into window
createTrackbar("H_MIN", "trackbars green", &H_MIN_green, 255, my_trackbars);
createTrackbar("H_MAX", "trackbars green", &H_MAX_green, 255, my_trackbars);
createTrackbar("S_MIN", "trackbars green", &S_MIN_green, 255, my_trackbars);
createTrackbar("S_MAX", "trackbars green", &S_MAX_green, 255, my_trackbars);
createTrackbar("V_MIN", "trackbars green", &V_MIN_green, 255, my_trackbars);
createTrackbar("V_MAX", "trackbars green", &V_MAX_green, 255, my_trackbars);

//create window for trackbars
namedWindow("Obstacles", WINDOW_AUTOSIZE);
//create memory to store trackbar name on window
char TrackbarNamepink[50];

//create trackbars and insert them into window
createTrackbar("H_MIN", "Obstacles", &H_MIN_pink, 255, my_trackbars);
createTrackbar("H_MAX", "Obstacles", &H_MAX_pink, 255, my_trackbars);
createTrackbar("S_MIN", "Obstacles", &S_MIN_pink, 255, my_trackbars);
createTrackbar("S_MAX", "Obstacles", &S_MAX_pink, 255, my_trackbars);
createTrackbar("V_MIN", "Obstacles", &V_MIN_pink, 255, my_trackbars);
createTrackbar("V_MAX", "Obstacles", &V_MAX_pink, 255, my_trackbars);
}

void track_object(int &color, int &x, int &y, Mat threshold, Mat &cameraFeed) {
    Mat temp;
    threshold.copyTo(temp);
    //these two vectors needed for output of findContours
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;
    //find contours of filtered image using openCV findContours function
    findContours(temp, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);
    //use moments method to find our filtered object
    double refArea = 0;
    bool objectFound = false;

```



```

if (hierarchy.size() > 0) {
    int numObjects = hierarchy.size(); //if this is big, we have a noisy image
    for (int index = 0; index <= 0; index = hierarchy[index][0]) {
        Moments moment = moments((cv::Mat)contours[index]);
        double area = moment.m00; //area
        x = moment.m10 / area; //this is where the proram finds the x/y coordinates of
middle of the object
        y = moment.m01 / area; //this is where the proram finds the x/y coordinates of
middle of the object
        objectFound = true;
        refArea = area;
    }
    //let user know you found an object
    if (objectFound == true) {
        putText(cameraFeed, "Tracking Object", Point(0, 50), 2, 1, Scalar(0, 255, 0), 2
        //draw object location on screen
        circle(cameraFeed, Point(x, y), 10, Scalar(0, 255, 0), 2);
        //pathfinding subroutine?
        object_found = 1;
        //printf("Color %d position: y = %d, z = %d, \n", color, x ,y);
    }
}
}

void track_obstacle(int &color, int &x, int &y, Mat threshold, Mat &cameraFeed) {
    Mat threshold_output;
    threshold.copyTo(threshold_output);
    //these two vectors needed for output of findContours
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;
    //find contours of filtered image using openCV findContours function
    findContours(threshold_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Poir
);

    // Approximate contours to polygons + get bounding rects and circles
    vector<vector<Point> > contours_poly(contours.size());
    vector<Rect> boundRect(contours.size());
    vector<Point2f>center( contours.size() );
    vector<float>radius(contours.size());
    int i;
    double refArea = 0;
    bool objectFound = false;
    if (hierarchy.size() > 0) {
        int numObjects = hierarchy.size(); //if this is big, we have a noisy image
        for (int index = 0; index <= 0; index = hierarchy[index][0]) {
            Moments moment = moments((cv::Mat)contours[index]);
            for (i = 0; i < contours.size(); i++) {
                approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
                boundRect[i] = boundingRect(Mat(contours_poly[i]));
                minEnclosingCircle((Mat)contours_poly[i], center[i], radius[i]);
            }
            double area = moment.m00; //area
            x = moment.m10 / area; //program finds coordinates of the middle of the object
            y = moment.m01 / area; //program finds coordinates of the middle of the object
            //int w = boundRect.width;
            //int h = boundRect.height;
            objectFound = true;
            refArea = area;
        }
    }
}

```

```

        //let user know you found an object
        if (objectFound == true) {
            for (i = 0; i < contours.size(); i++) {
                rectangle(cameraFeed, boundRect[i].tl(), boundRect[i].br(), Scalar(0, 25
2, 8, 0)); //tl == top left, br == bottom right
                circle(cameraFeed, center[i], (int)radius[i], Scalar(255, 0, 0), 2, 8, 0
            }
            putText(cameraFeed, "Tracking Obstacle", Point(0, 50), 2, 1, Scalar(0, 255, 0),
            circle(cameraFeed, Point(x, y), 10, Scalar(255, 0, 0), 2);
            object_found = 1;
        }
    }
}

//experimental python interpreter. remove this section if breaking code
string Call_Pathfinder_PyObject()
{
    Py_Initialize();

    // Create some Python objects that will later be assigned values.
    PyObject *pName, *pModule, *pDict, *pFunc, *pArgs, *pObstacles, *pWidth, *pHeight, *pDirs, *p
    *pxB, *pyA, *pyB;
    pName = PyString_FromString("star"); // Convert the file name to a Python string.
    pModule = PyImport_Import(pName); // Import the file as a Python module.
    pDict = PyModule_GetDict(pModule); // Create a dictionary for the contents of the modul
    pFunc = PyDict_GetItemString(pDict, "pathFind"); // Get the add method from the dictionary.
    pArgs = PyTuple_New(7); // Create a Python tuple to hold the arguments to the method
    pWidth = PyInt_FromLong(MAP_WIDTH);
    pHeight = PyInt_FromLong(MAP_HEIGHT);
    pDirs = PyInt_FromLong(Direction);
    pxA = PyInt_FromLong(89);
    pyA = PyInt_FromLong(5);
    pxB = PyInt_FromLong(7);
    pyB = PyInt_FromLong(48);

    //so were gonna send a string to python
    // Set the Python int as the first and second arguments to the method.

    PyTuple_SetItem(pArgs, 0, pWidth);
    PyTuple_SetItem(pArgs, 1, pHeight);
    PyTuple_SetItem(pArgs, 2, pDirs);
    PyTuple_SetItem(pArgs, 3, pxA);
    PyTuple_SetItem(pArgs, 4, pyA);
    PyTuple_SetItem(pArgs, 5, pxB);
    PyTuple_SetItem(pArgs, 6, pyB);

    // Call the function with the arguments.
    PyObject* pResult = PyObject_CallObject(pFunc, pArgs);

    if (pResult == NULL)
        printf("Calling the add method failed.\n");
    // Convert the result to a string from a Python object.
    string route = PyString_AsString(pResult); //Destroy the Python interpreter.
    Py_Finalize();
    return route;
}

```

```

int main()
{
    Mat cameraFeed;
    Mat HSV;
    Mat pink_threshold;
    Mat green_threshold;
    Mat red_threshold;
    Mat blue_threshold;
    //x and y values for the location of the object

    int color = 0;
    createTrackbars();
    VideoCapture capture;
    capture.open(camera_number);

    string GetRoute = Call_Pathfinder_PyObject();    //get python path string

    int first_loop = 0; //if the first loop, position the windows, don't do it over and over or I
    not be able to drag windows around
    while (1) {
        int rx = 0, ry = 0; //red x and y
        int gx = 0, gy = 0; //green x and y
        int bx = 0, by = 0; //blue x and y
        int px = 0, py = 0; //pink x and y
        capture.read(cameraFeed);
        resize(cameraFeed, cameraFeed, Size(400, 300));
        //convert frame from BGR to HSV colorspace
        cvtColor(cameraFeed, HSV, COLOR_BGR2HSV);
        //filter and store in threshold matrix

        inRange(HSV, Scalar(H_MIN_green, S_MIN_green, V_MIN_green), Scalar(H_MAX_green, S_MAX_
V_MAX_green), green_threshold);
        //erode and dilate to clean up the image
        Mat erodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
        Mat dilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));
        erode(green_threshold, green_threshold, erodeElement);
        dilate(green_threshold, green_threshold, dilateElement);
        //track the object
        color = 1;
        track_object(color, gx, gy, green_threshold, cameraFeed);

        inRange(HSV, Scalar(H_MIN_blue, S_MIN_blue, V_MIN_blue), Scalar(H_MAX_blue, S_MAX_blue
V_MAX_blue), blue_threshold);
        //erode and dilate to clean up the image
        Mat erodeElement_blue = getStructuringElement(MORPH_RECT, Size(3, 3));
        Mat dilateElement_blue = getStructuringElement(MORPH_RECT, Size(8, 8));
        erode(blue_threshold, blue_threshold, erodeElement);
        dilate(blue_threshold, blue_threshold, dilateElement);
        //track the object
        color = 2;
        track_object(color, bx, by, blue_threshold, cameraFeed);

        inRange(HSV, Scalar(H_MIN_red, S_MIN_red, V_MIN_red), Scalar(H_MAX_red, S_MAX_red, V_M
red_threshold);
        //erode and dilate to clean up the image
        Mat erodeElement_red = getStructuringElement(MORPH_RECT, Size(3, 3));

```

```

    Mat dilateElement_red = getStructuringElement(MORPH_RECT, Size(8, 8));
    erode(red_threshold, red_threshold, erodeElement);
    dilate(red_threshold, red_threshold, dilateElement);
    //track the object
    color = 2;
    track_object(color, rx, ry, red_threshold, cameraFeed);

    inRange(HSV, Scalar(H_MIN_pink, S_MIN_pink, V_MIN_pink), Scalar(H_MAX_pink, S_MAX_pink,
V_MAX_pink), pink_threshold);
    //erode and dilate to clean up the image
    Mat erodeElement_pink = getStructuringElement(MORPH_RECT, Size(3, 3));
    Mat dilateElement_pink = getStructuringElement(MORPH_RECT, Size(8, 8));
    erode(pink_threshold, pink_threshold, erodeElement);
    dilate(pink_threshold, pink_threshold, dilateElement);
    //erode(pink_threshold, pink_threshold, erodeElement);
    //dilate(pink_threshold, pink_threshold, dilateElement);
    //erode(pink_threshold, pink_threshold, erodeElement);
    //dilate(pink_threshold, pink_threshold, dilateElement);
    //blur(HSV, HSV, Size(3, 3));
    //track the object
    color = 2;
    track_obstacle(color, px, py, pink_threshold, cameraFeed);

    //get distance between points
    float a, b, dist_gp;
    a = gx - px;
    b = gy - py;
    dist_gp = sqrt((a*a) + (b*b));

    float c, d, dist_gr;
    c = gx - rx;
    d = gy - ry;
    dist_gr = sqrt((c*c) + (d*d));

    //get angle between points
    float angle_gp = 0;

    if (a == 0 && b != 0) { // on x-axis
        angle_gp = (b < 0) ? 180 : 0;
    }
    else if (a != 0 && b == 0) { // on y-axis
        angle_gp = (a < 0) ? -90 : 90;
    }
    else { // normal move
        if (b > 0) { // correction factor to scale with OpenCV headings
            angle_gp = 0;
        }
        else {
            angle_gp = (a > 0) ? 180 : -180;
        }
        //printf("b = %f\n", b);
        angle_gp += (atan(a / b) * 180.0) / 3.141592654;
        angle_gp -= (angle_gp > 180) ? 360 : 0;
        angle_gp += (angle_gp < -180) ? 360 : 0;
    }
    /*
    if (b != 0)
        angle_gp = atan(a / b);
    angle_gp = (angle_gp * 180) / 3.141592654;

```

```

*/

float angle_gr = 0;
if (d != 0)
    angle_gr = atan(c / d);
angle_gr = (angle_gr * 180) / 3.141592654;

//cout << "\n The hypotenuse length is: " << a << " " << b << " " << (result);

printf("GRN:(%d,%d)\tPNK:(%d,%d)\tRED:(%d,%d)\tBLU:(%d,%d)\tG-P:(%d,%d)\tAngGP:%.1f%c\tDstGP:%d\n", gx, gy, px, py, rx, ry, bx, by, gx - px, gy - py, angle_gr, 248, dist_gp, rx - bx, ry - by, angle_gr, 248, dist_gr); //subtrace pink coordinates from pink the delta
ofstream my_file;
//write a file with motor commands
if (object_found == 1) {
    try {
        my_file.open("../cat_tracking.txt");
        if (!my_file.is_open())
            throw 1;
        my_file << "greenX " << gx << "\n";
        my_file << "greenY " << gy << "\n";
        my_file << "pinkX " << px << "\n";
        my_file << "pinkY " << py << "\n";
        //my_file << "catAng "; //y
        //my_file << angle_gr;
        //my_file << "\n";
        my_file << "dot1X ";
        my_file << bx;
        my_file << "\n";
        my_file << "dot1Y "; //y
        my_file << by;
        my_file << "\n";
    }
    catch (int e)
    {
    }
    my_file.close();
}
//open all windows
imshow("camera_feed", cameraFeed);
imshow("green_threshold", green_threshold);
imshow("pink_threshold", pink_threshold);
imshow("red_threshold", red_threshold);
imshow("blue_threshold", blue_threshold);
if (first_loop == 0) {
    first_loop = 1;
    moveWindow("camera_feed", 600, 330);
    moveWindow("green_threshold", 0, 0);
    moveWindow("pink_threshold", 430, 0);
    moveWindow("red_threshold", 860, 0);
    moveWindow("blue_threshold", 1290, 0);
    resizeWindow("trackbars green", 300, 200);
    resizeWindow("trackbars pink", 300, 200);
    resizeWindow("trackbars red", 300, 200);
    resizeWindow("trackbars blue", 300, 200);
    moveWindow("trackbars green", 0, 500);
    moveWindow("trackbars pink", 430, 500);
    moveWindow("trackbars red", 860, 500);
}

```

```

        moveWindow("trackbars blue", 1290, 500);
    }
    waitKey(40);
}
}

```

PathFinding Star.py

```

from heapq import heappush, heappop # for priority queue
import math
import time
import random
import socket
class node:
    xPos = 0 # x position
    yPos = 0 # y position
    distance = 0 # total distance already travelled to reach the node
    priority = 0 # priority = distance + remaining distance estimate
    def __init__(self, xPos, yPos, distance, priority):
        self.xPos = xPos
        self.yPos = yPos
        self.distance = distance
        self.priority = priority
    def __lt__(self, other): # comparison method for priority queue
        return self.priority < other.priority
    def updatePriority(self, xDest, yDest):
        self.priority = self.distance + self.estimate(xDest, yDest) * 10 # A*
# give higher priority to going straight instead of diagonally
    def nextMove(self, dirs, d): # d: direction to move
        if dirs == 8 and d % 2 != 0:
            self.distance += 14
        else:
            self.distance += 10
# Estimation function for the remaining distance to the goal.
    def estimate(self, xDest, yDest):
        xd = xDest - self.xPos
        yd = yDest - self.yPos
        # Euclidian Distance
        d = math.sqrt(xd * xd + yd * yd)
        # Manhattan distance
        # d = abs(xd) + abs(yd)
        # Chebyshev distance
        # d = max(abs(xd), abs(yd))
        return(d)
# A-star algorithm.
# The path returned will be a string of digits of directions.
    def pathFind(n, m, dirs, xA, yA, xB, yB):#, obstacle_string):
#global the_map
    dx = [1, 1, 0, -1, -1, -1, 0, 1]
    dy = [0, 1, 1, 1, 0, -1, -1, -1]
    the_map = []
    closed_nodes_map = [] # map of closed (tried-out) nodes

```

```

open_nodes_map = [] # map of open (not-yet-tried) nodes
dir_map = [] # map of dirs
row = [0] * n
for i in range(m): # create 2d arrays
    closed_nodes_map.append(list(row))
    open_nodes_map.append(list(row))
    dir_map.append(list(row)) # create empty map
    the_map.append(list(row))
obstacle_string = "12,34,6;23,43,8;23,65,9"
#print obstacle_string
#write obstacles here
obstacle_list = obstacle_string.split(';') # Break string into two lists

for i in range(len(obstacle_list)): # Access individual elements within the obstacle list
    obs_list = obstacle_list[i].strip()
    item = obs_list.split(',')
    print item
    #for j in item:
        #print j.strip
        DrawCircle(int(item[0]),int(item[1]),int(item[2]),n,m, the_map) #mark the map
pq = [[], []] # priority queues of open (not-yet-tried) nodes
pqi = 0 # priority queue index
# create the start node and push into list of open nodes
n0 = node(xA, yA, 0, 0)
n0.updatePriority(xB, yB)
heappush(pq[pqi], n0)
open_nodes_map[yA][xA] = n0.priority # mark it on the open nodes map

# A* search
while len(pq[pqi]) > 0:
    # get the current node w/ the highest priority
    # from the list of open nodes
    n1 = pq[pqi][0] # top node
    n0 = node(n1.xPos, n1.yPos, n1.distance, n1.priority)
    x = n0.xPos
    y = n0.yPos
    heappop(pq[pqi]) # remove the node from the open list
    open_nodes_map[y][x] = 0
    closed_nodes_map[y][x] = 1 # mark it on the closed nodes map
    # quit searching when the goal is reached
    # if n0.estimate(xB, yB) == 0:
    if x == xB and y == yB:
        # generate the path from finish to start
        # by following the dirs
        path = ''
        while not (x == xA and y == yA):
            j = dir_map[y][x]
            c = str((j + dirs / 2) % dirs)
            path = c + path
            x += dx[j]
            y += dy[j]

        # mark the route on the map
        if len(path) > 0:
            x = xA
            y = yA
            the_map[y][x] = 2
            for i in range(len(path)):

```

```

        j = int(path[i])
        x += dx[j]
        y += dy[j]
        the_map[y][x] = 3
        #print 'currnt direction:' , j
        #CreateCommandTarget(j,CommandDelegate,CommandList)
the_map[y][x] = 4

print 'Map:'
for y in range(m):
for x in range(n):
    xy = the_map[y][x]
    if xy == 0:
        print '.', # space
    elif xy == 1:
        print 'O', # obstacle
    elif xy == 2:
        print 'S', # start
    elif xy == 3:
        print 'R', # route
    elif xy == 4:
        print 'F', # finish

print

    return path
# generate moves (child nodes) in all possible dirs
for i in range(dirs):
    xdx = x + dx[i]
    ydy = y + dy[i]
    if not (xdx < 0 or xdx > n-1 or ydy < 0 or ydy > m - 1
            or the_map[ydy][xdx] == 1 or closed_nodes_map[ydy][xdx] == 1):
        # generate a child node
        m0 = node(xdx, ydy, n0.distance, n0.priority)
        m0.nextMove(dirs, i)
        m0.updatePriority(xB, yB)
        # if it is not in the open list then add into that
        if open_nodes_map[ydy][xdx] == 0:
            open_nodes_map[ydy][xdx] = m0.priority
            heappush(pq[pqi], m0)
            # mark its parent node direction
            dir_map[ydy][xdx] = (i + dirs / 2) % dirs
        elif open_nodes_map[ydy][xdx] > m0.priority:
            # update the priority
            open_nodes_map[ydy][xdx] = m0.priority
            # update the parent direction
            dir_map[ydy][xdx] = (i + dirs / 2) % dirs
            # replace the node
            # by emptying one pq to the other one
            # except the node to be replaced will be ignored
            # and the new node will be pushed in instead
            while not (pq[pqi][0].xPos == xdx and pq[pqi][0].yPos == ydy):
                heappush(pq[1 - pqi], pq[pqi][0])
                heappop(pq[pqi])
            heappop(pq[pqi]) # remove the target node

# empty the larger size priority queue to the smaller one
if len(pq[pqi]) > len(pq[1 - pqi]):
    pqi = 1 - pqi
    while len(pq[pqi]) > 0:
        heappush(pq[1-pqi], pq[pqi][0])

```



```

        heappop(pq[pqi])
        pqi = 1 - pqi
        heappush(pq[pqi], m0)
        # add the better node instead
        # if no route found
    return ''
#Draw circles based on start x and y of center and radius
def DrawCircle(StartX, StartY, radius, n, m, the_map):
    for i in range (StartX - radius, StartX + radius):
        for j in range (StartY - radius, StartY + radius):
            if ((i - StartX)*(i - StartX) + (j - StartY)*(j - StartY) <= radius * radius and ((i >= 0)
            >= 0) and (i <= n-1) and (j <= m-1))):
                the_map[i][j] = 1

#UDP client code for sending command
def Client_send(msm_string):
    host = '127.0.0.1'
    port = 3100
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.sendto(msm_string + "\n", (host, port))
    s.close()
def CreateCommandTarget(Dir, Com_Del, Com_List):
    global count
    msm = ""
    if not Com_Del:
        Com_Del.append(Dir)
        return
    elif len(Com_Del) == 1:
        Com_Del.insert(0, Dir)
    else:
        Com_Del.insert(0, Dir)
        Com_Del.pop(2)
    print Com_Del
    c = Com_Del[0]
    p = Com_Del[1]
    if ( c == p ):
        count += 1
        msm = "1"
        print "MoveForward"
    elif ( p - c == 1 and c != 0 ):
        count += 1
        print "Turn left 45, Move forward"
        msm = "2 45"
    elif ( p - c == 2 and c != 0 ):
        count += 1
        print "Turn left 90, Move forward"
        msm = "2 90"
    elif ( p - c == 3 and c != 0 ):
        count += 1
        print "Turn left 135, Move forward"
        msm = "2 135"
    elif ( c - p == 1 and c != 0 ):
        count += 1
        print "Turn right 45, Move forward"
        msm = "2 -45"
    elif ( c - p == 2 and c != 0 ):
        count += 1
        print "Turn right 90, Move forward"
        msm = "2 -90"
    elif ( c - p == 3 and c != 0 ):
        count += 1
        print "Turn right 135, Move forward"

```

```

        msm = "2 -135"
    elif ( c == 0 and p == 7) or (p == 0 and c == 1):
        count += 1
        print "Turn right 45, Move forward"
        msm = "2 -45"
    elif ( c == 0 and p == 6) or (p == 0 and c == 2):
        count += 1
        print "Turn right 90, Move forward"
        msm = "2 -90"
    elif ( c == 0 and p == 5) or (p == 0 and c == 3):
        count += 1
        print "Turn right 135, Move forward"
        msm = "2 -135"
    elif ( c == 0 and p == 1) or (p == 0 and c == 7):
        count += 1
        print "Turn left 45, Move forward"
        msm = "2 45"
    elif ( c == 0 and p == 2) or (p == 0 and c == 6):
        count += 1
        print "Turn left 90, Move forward"
        msm = "2 90"
    elif ( c == 0 and p == 3) or (p == 0 and c == 5):
        count += 1
        print "Turn left 135, Move forward"
        msm = "2 135"
    print msm
    Client_send(msm)                                     #send commands to UDP delegation

#def draw_map(the_map,m,n):

# MAIN
def Main():
    global count
    global dx
    global dy
    global the_map
    count = 0
    dirs = 8                                             # number of possible directions
    on the map
    dx = [1, 1, 0, -1, -1, -1, 0, 1]
    dy = [0, 1, 1, 1, 0, -1, -1, -1]

    # 0 = right, 1 = down right , 2 = down, 3 = down-left , 4 = left, 5 = up-left, 6 = up, 7 = Up-right
    n = 100                                             # horizontal size of the map
    m = 100                                             # vertical size of the map
    the_map = []
    row = [0] * n
    CommandDelegate = []
    CommandList = ['MOVE_FORWARD', 'MOVE_TURNLEFT', 'MOVE_TURNRIGHT', 'MOVE_BACK', 'WALK_READY', 'WALK_SLOW',
                    'WALK_FAST']

    # Defined start location for robot and destination
    global xA
    global yA
    (xA, yA) = (89, 5)                                # robot position
    (xB, yB) = (7, 48)                                # Final Destination
    print 'Map size (X,Y): ', n, m
    print 'Start: ', xA, yA
    print 'Finish: ', xB, yB

```

```

#while (xA != xB and yA != yB):
# Actual Running A Star Algorithm

t = time.time()
string_list = "12,34,6;23,43,8;23,65,9"
route = pathFind(n, m, dirs, xA, yA, xB, yB)
print 'Time to generate the route (seconds): ', time.time() - t
print 'Route:'
print route

if __name__=="__main__":
    Main()

```