eYSIP−2018

A system for solving jigsaw puzzle using multiple robots

Aniket Anantraj Navlur
Kiran Suvas Patil
Ashis Kumar Maharana
Mentor 1: Abinav Sarkar
Mentor 2: Kalind Karia
Duration of Internship: 21/05/2018 – 06/07/2018

2018, e-Yantra Publication

A System for Solving Jigsaw Puzzle using Multiple Robots

Abstract

The prime motive of this project is to develop a multi Robot based Autonomous Puzzle Solver system that can solve a Jigsaw puzzle.

Overview of Task

Task No.	Task
1	Python, OpenCV, Firebird V Intro,XBee Communication
2	Pose and orientation calculation of 2 Firebird robots using color/Aruco markers
3	Programming the Go-To-Goal Controller for single Firebird V robot. Tuning the PID values to perfection
4	Implementing path planning with Firebird V where obstacles have been placed in arena
5	Detection of jigsaw puzzle blocks usingTemplate Matching
6	Pick and place of blocks - gripper mechanism building
7	Implementing the entire solution for a givenjigsaw puzzle
8	Documentation and reporting results

Completion status

Whole of the solution is successfully implemented using a single robot.

1.1 Hardware parts

- FireBird Robots(2)

- XBee(3)
  - Xbee module
  - Xbee usb adapter board
  - USB type B to type A converter
- Overhead Camera
- ArUco markers
- Servo motors for gripper mechanism
- Flex sheet for arena
- Puzzle pieces

1.2 Software used

- Python
- OpenCV
- AVR Studio 5.1
- XCTU(latest version)
- AVRDude
- Fusion360

1.3 Hardware Design

Gripper Mechanism

The gripper mechanism is build for a bigger block size $12cm \times 12cm \times 12cm$. Alongside the gripper we need two more degrees of freedom. One for picking the block and another for rotating the block. As the block has to be rotated around its z-axis we are picking it from top. That makes the gripper to be placed at a height higher than $12cm$. And the servo that will rotate the block has to be placed higher than that and at some distance such that while rotating or opening the gripper, it should not touch the robot in order not to damage the gripper. We are using dual shaft servos and their brackets. Once

the platform and arm is made we went for some inspiration from the internet for the gripper and tried simulating different ones. Then we finished with a simpler one not to make it heavy for the arm and servo. Once designed in *Fusion*360 and tested in simulation, we 3D printed the prototype. Once satisfied with the performance we printed two more of them for two robots. For a good grip we made the gripping plates wider and are using rubber bands and foam. By changing the gripping angle it can pick a block as big as $12cm$.
The final gripper that is designed is given below. All these parts are designed in **Fusion 360** with proper measurements and 3D printed.

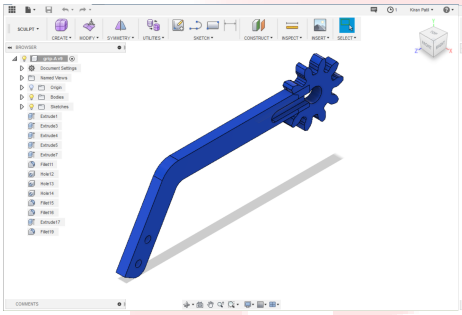

Figure. **Right Gear**

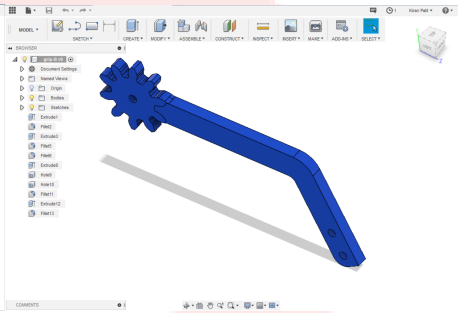

Figure. **Left Gear**

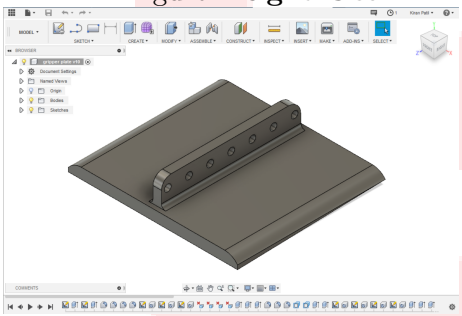

Figure. **Gripping Plate**

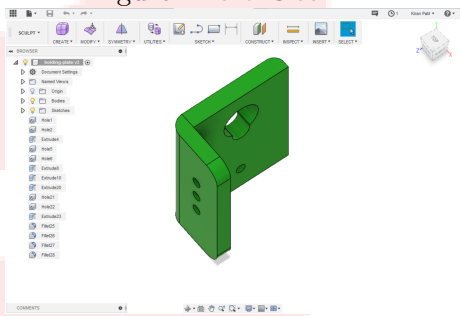

Figure. **Servo Holder**

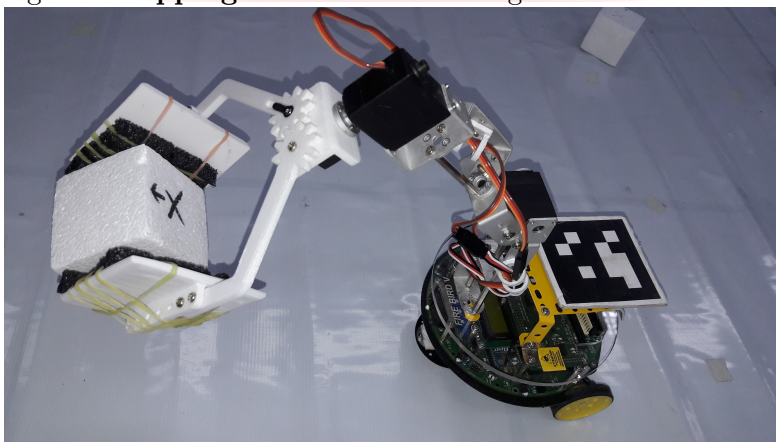

Figure. **FInal assembled arm and gripper**

1.4 Software and Code

Dependencies

Before configuring the XBee modules one should first understand the different parameters of it. This article[1] should help. For configuring the xbee modules latest XCTU software is used. The channel is chosen so, such that no other device interfere the communication.

>>ArUco module

Detection of ArUco markers using python requires integration of aruco library to the current python version installed. The aruco library can be found in opencv-contrib module. For installing opencv-contrib follow these steps[2] (installing modules using pip).

```
pip install opencv-contrib-python
```

>>Set up for Communication

For communicating to xbee module using python requires xbee and pyserial module. There are many similar modules for serial communication and xbee, but we need these two modules. One can search and go through other modules also. One is given below.

1. pyserial[3]

```
pip install pyserial
```

2. xbee[4]

```
pip install xbee
```

3. optional digi-xbee[5]

```
pip install digi-xbee
```

The xbee model provided(XBee S2C) cannot be read properly using the earlier versions of XCTU. So it is recommended to use the latest version[6]. Follow the previous Docs for configuration using XCTU. After configuration and testing over XCTU, communication is again tested using the python script. Once the communication is established it is time to connect the xbee modules on the Firebird V robots. How to guide can be found in the hardware manual of Firebird V.

>>**Image rotation**

For rotating the image in case of template matching we are using rotate function available in imutils module. To install imutils[7]

```
pip install imutils
```

ArUco detection

Then comes the localisation of robots. For that purpose we are using overhead camera and ArUco markers. The position and orientation of the ArUco marker is first calculated using python script and math module[8] which is inbuilt in python. Angle is calculated such that it ranges from 0 to 360 degrees. The python script can be found here.

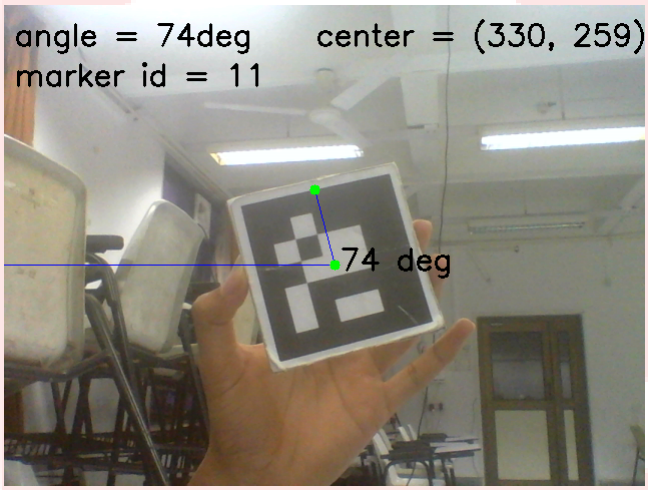

Figure. **ArUco orientation and position**

Go-To-Goal Controller

For the Go-To-Goal Controller we first found the orientation and distance of the robot from its goal or destination. Then the error angle is calculated by subtracting angle with respect to frame from angle with respect to destination. The code can be found here. In another way the error angle can be found by

$$errorangle = \arctan\left(\frac{m-n}{1-m\times n}\right)$$

m = orientation of robot with respect to frame
n = orientation of robot with respect to destination

This error angle is then passed to the PID controller which then adjusts the speed of motors to maintain a zero error. The python script sends data packets to the Firebird V robot via Xbee. These data packets are formed by the following values.

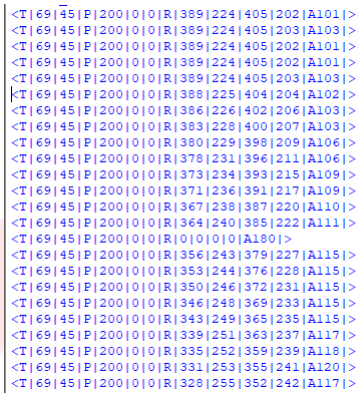

Figure. **Packet example**

$\langle T|d_x|d_y|P|K_p|K_i|K_d|R|r_x|r_y|A|angle|\rangle$

< symbolizes beginning of data packet

T symbolizes target

d_x = destination x co-ordinate

d_y = destination y co-ordinate

P symbolizes PID gains

K_p = proportional gain

K_i = integral gain

K_d = differential gain

R symbolizes robot position

r_x = robot x co-ordinate

r_y = robot y co-ordinate

A symbolizes error angle

> symbolizes end of packet

Once received, the packet is successfully parsed by the robot in an ISR and the speeds are assigned to the motors as PWM duty-cycle.The c code and python script can be found here. When the robot is around 10 pixels away from its destination it stops, showing it has reached the destination. This is because the robot is bigger then the marker and before the center point reaches its destination robot will have already reached there. We completed the path planning in various steps. When the destination is given to the python script it first determines which of the two robots is nearer to the destination and then it sends packets to that robot. Video can be found here.

Path planning with Obstacle avoidance

The next step is to avoid obstacles. For a robot moving towards its destination, the other robot and the puzzle blocks except the one at its destination are the obstacles for the robot. One of the obstacles is a moving one so the obstacle map needs to be dynamic. So we have to build the obstacle map in every frame. And the path finding algorithm should be fast enough too. We tried various algorithms like dijkstra's[10] with voronoi[9], Lee's, A*[11] and finally concluded with A* algorithm which seemed fast and reliable. The center points of the obstacles are given to a function which then draws a rectangle around it. The area of the rectangle is treated as the obstacle. So the path finding algorithm avoids those rectangles. We have taken care of the size of the robot by drawing circle around the center of the robot. The path is planned such that there is some distance between the robot's outer shell and the obstacle. Robot's body should not touch the obstacle. First it is done for a single robot then it is implemented for a multi robot environment. Video can be found here.

Template Matching

For solving a jigsaw puzzle we first have to determine which puzzle block goes where. This is done by template matching. We are given with the static image of the arena where there are puzzle blocks placed with different orientation and the final image we have to make out of those puzzle blocks. We first detect the puzzle block from the arena and separate it from background. The final image is divided into same number of pieces as present in the arena. We number each of the piece according to their position. Then we run a template matching function which rotate the block obtained from the arena by 1 degree and finds percentage matching with every piece of final image. For which number piece the matching percentage is maximum, is then assigned to the block along with the angle that gave the maximum match. The code can be found here. This angle and robot's orientation both play major role in picking and placing the puzzle block at its destination.

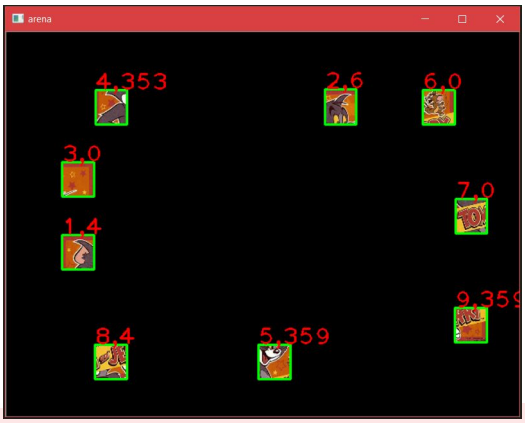

Figure. **Puzzle block no. and orientation obtained from Template matching**

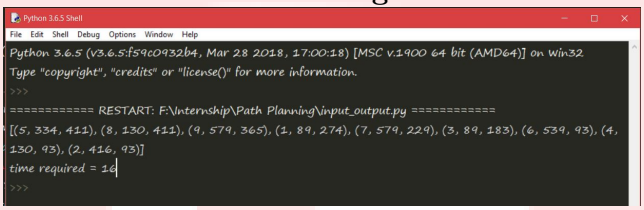

Figure. **Puzzle block no. and block position**

Final Implementation

The whole of the solution is first implemented for a single robot. Every block has 4 pickup points around it and at the center there is a point assigned where it will be placed and where the robot has to be to place it correctly. The path finding algorithm finds the nearest point out of those 4 point for the nearest block and goes there. Once reached it then aligns parallel to the side of block it is facing. It then rotates the gripper if needed and picks the block, travel to the destination for that particular block avoiding all other blocks, place it in proper orientation and then moves towards next nearest block.

1.5 Demo

Youtube Link of demonstration video

1.6 Future Work

A multi robot box pushing problem can be implemented by tweaking some of the functions and using complex mathematics. It will be tough.

1.7 Bug report and Challenges

- ArUco orientation
  The math module and the math header file in python and c respectively includes *arctan*, but for slope greater than -1 and less than 1 it does not give any output. We are unable to get the the angles between -45 and 45. So we went for *arccot* for this range then converted it back into *arctan*. So we are calculating the angle and sending it to the robot instead of letting it calculate.

- Block size
  When building the obstacle map it is taken care that the grippers of two robots do not collide.

- Gripper
  The gripper mechanism needs to be light and big enough to pick and place a big size block. Light because it should not add extra weight to the servo.

- Moving obstacle
  Path planning has to be done avoiding the moving obstacle.

- Camera
  The frame captured from the camera should be big enough to accommodate all the blocks and two robots and also focus should be good for the markers to be detected properly everywhere.

Bibliography

[1] Exploring xbees and xctu
https : *//learn.sparkfun.com/tutorials/exploring* − *xbees* − *and* − *xctu*

[2] installing modules using pip
https : *//www.youtube.com/watch?v* = *jnpC*$_I$*b*$_l$*bc*

[3] pyserial library
https : *//pypi.org/project/pyserial/*

[4] XBee library
https : *//pypi.org/project/XBee/*

[5] digi-xbee library
https : *//pypi.org/project/digi* − *xbee/*

[6] Download link XCTU latest version
https : *//www.digi.com/support/productdetail?pid* = 3352&*type* = *utilities*

[7] imutils library
https : *//www.pyimagesearch.com*/2015/02/02/*just*−*open*−*sourced*−*personal*−*imutils*−*package*−*series*−*opencv*−*convenience*−*functions/*

[8] math library
https : *//docs.python.org*/2/*library/math.html*

[9] Gripper Design inspiration
https : *//www.thingiverse.com/search?q* = *gripper* + *arm*&*dwh* = 985*b*3*b*60*dfeacfe*

[10] voronoi
https : *//docs.scipy.org/doc/scipy/reference/spatial.html*

[11] Dijkstra's algorithm
https : *//gist.github.com/econchick*/4666413

[12] A* path finding algorithm explanation
https : *//www.youtube.com/watch?v* = *aKYlikFAV*4*k&t* = 1429*s*

[13] A* path finding algorithm python
http : *//code.activestate.com/recipes/*578919 − *python* − *a* − *pathfinding* − *with* − *binary* − *heap/*