

Contents

1. Connection with Raven Software

Here is the big picture of how this program is going to interact with the main RAVEN software. Basically we are subscribing to ROS topic “ravenstate” to retrieve robot current status then publish to ROS topic “raven_automove”. We also include the UDP network packet method that raven_absolute_controller by Imperial College London use to connect with RAVEN. These two are very different approaches which is why we didn’t end up implementing our idea on their code.

2. AutoCircle generater Design Hierarchy

In our code, there are two main classes, the first one is Raven_Controller, it contains the two threads – ros_process and console_process that this program is running; the other class is Raven_PathPlanner, this class does all the math and computes circular trajectory.

3. Circle Trajectory Algorithms

Throughout the developing process, we tried four algorithms of circle trajectory implementation, here is the graphic explanation of each algorithm and brief description of the pros and cons, and why we end up using or not using them.

4. Regulating Motion Around Desired Radius

The key part of circular trajectory planning is to ensure the robot to move around the desired radius. But strict regulations can cause shaky motion so as long as the radius converges to a certain value, we would not push too hard on forcing it to be exactly at our desired radius. Here is our design diagram.

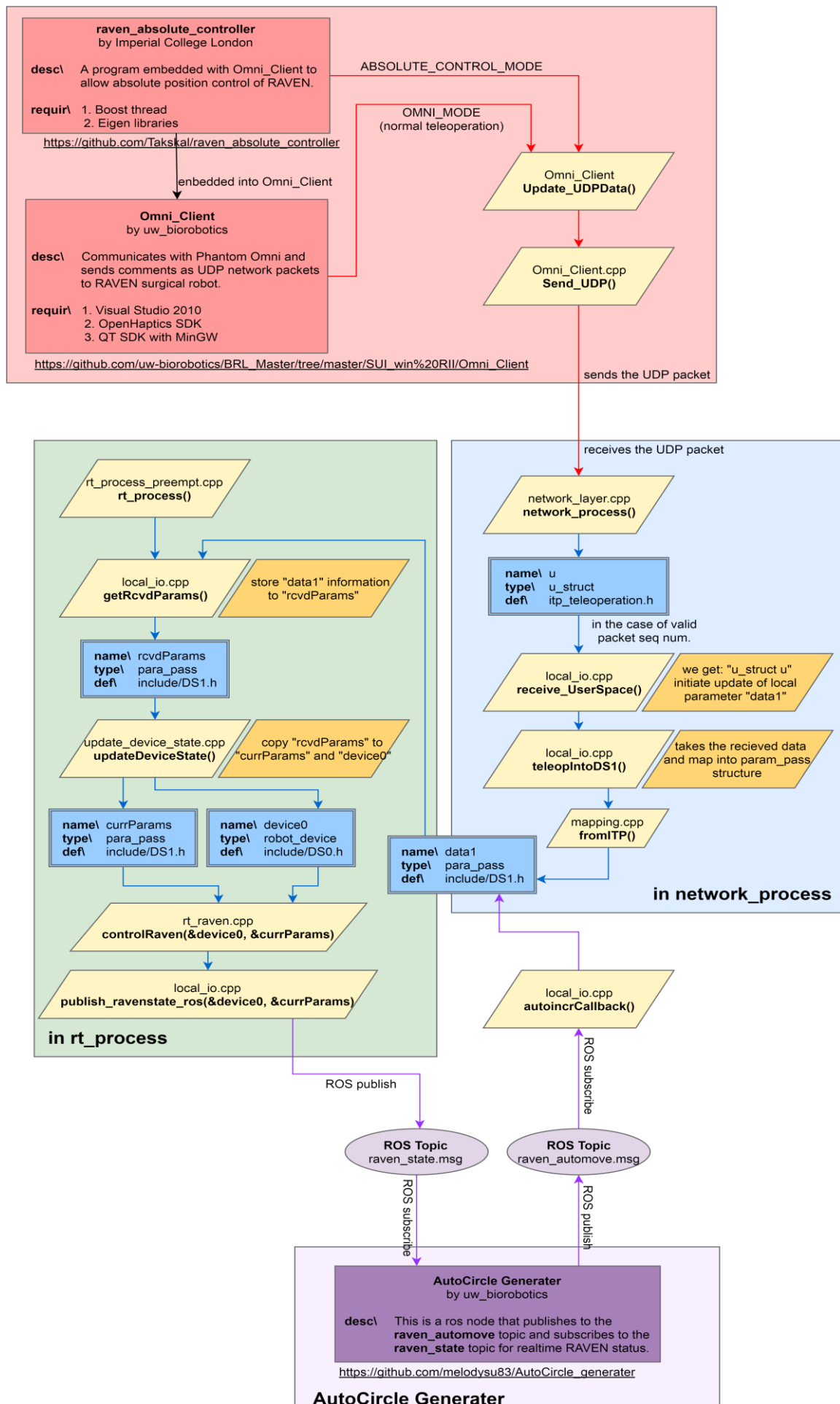
5. Demo Screenshots of AutoCircle generater

Here are some screenshots of how it looks like when our program actually runs. The first photo is the main selection menu for user to choose from, and the second one is the ROS publish/subscribe status display.

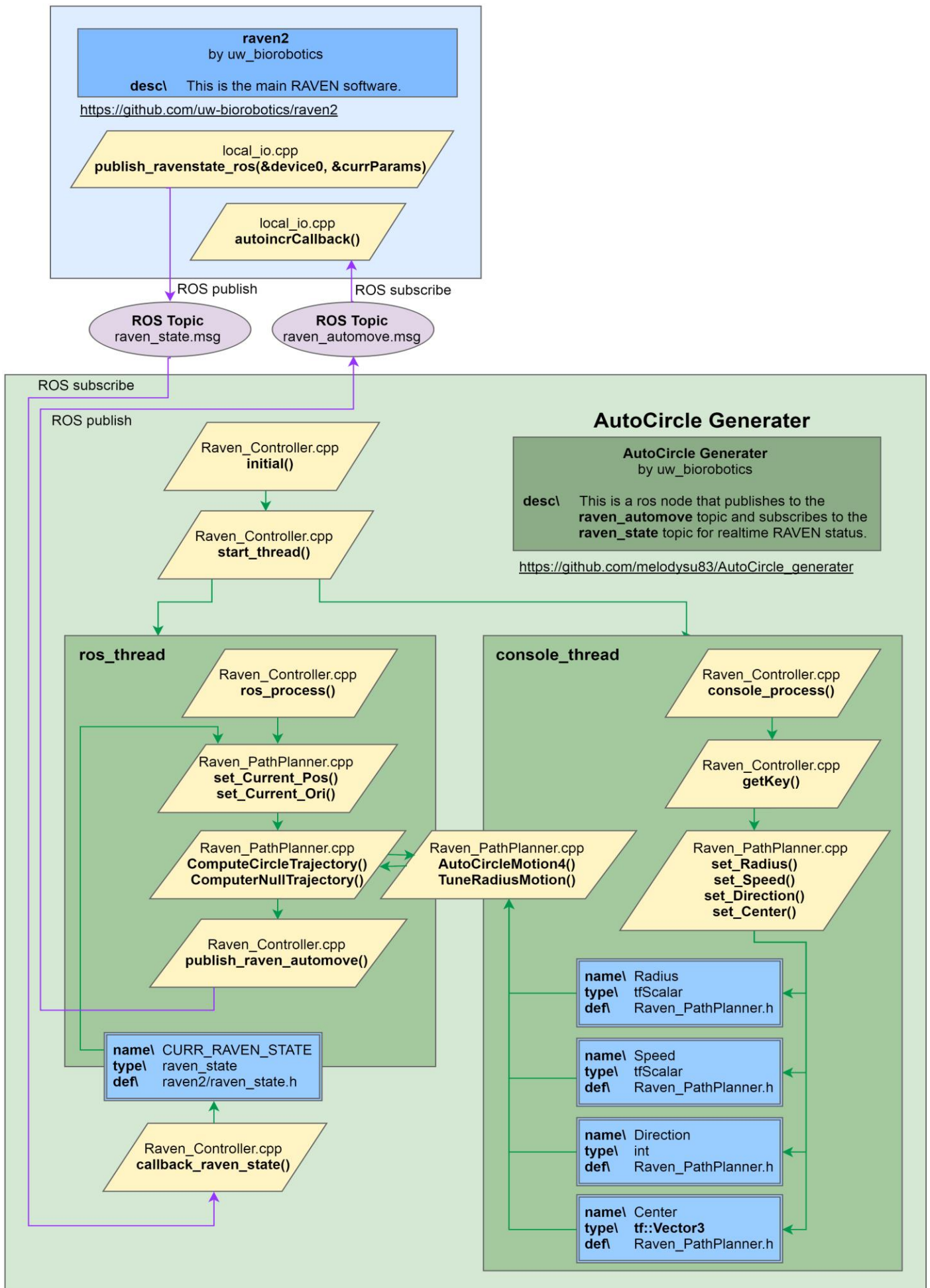
6. Data Analysis on Matlab

Here are some run time data we extracted when executing our AutoCircle generater program.

1. Connection with Raven Software



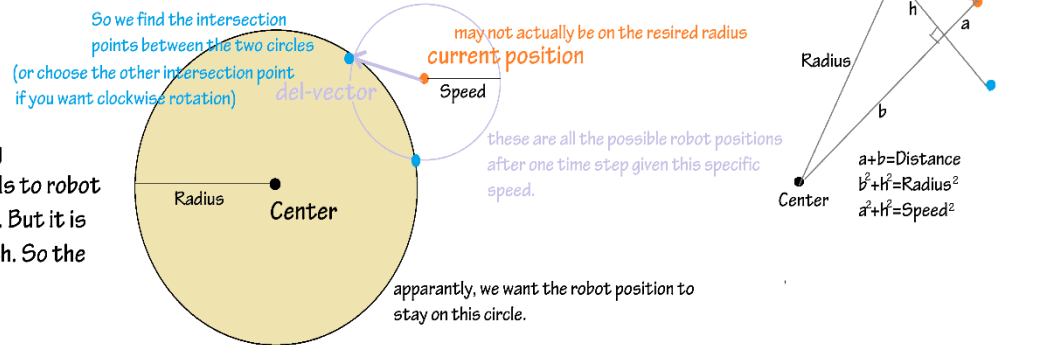
2. AutoCircle generater Design Hierarchy



3. Circle Trajectory Algorithms

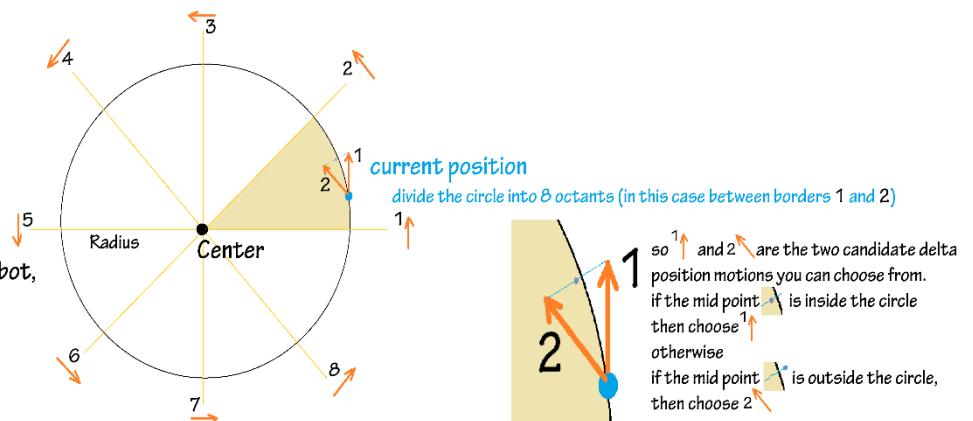
AutoCircleMotion1

The most precise way of generating commands that closely corresponds to robot current position, radius, and speed. But it is very not constant and not smooth. So the outcome was pretty unstable.



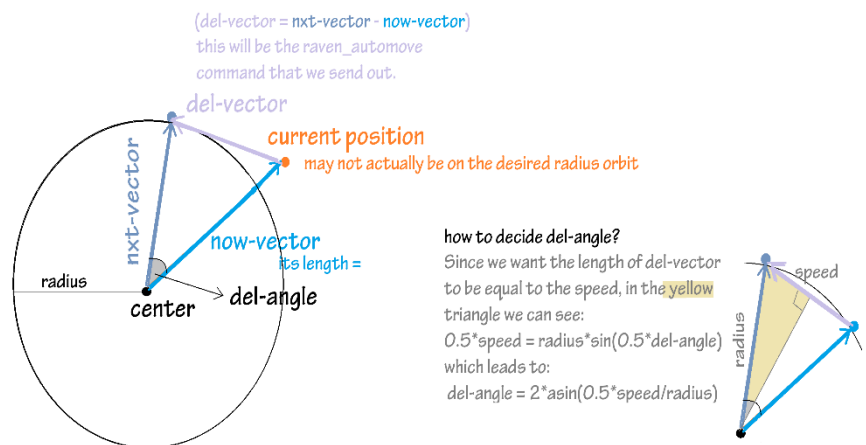
AutoCircleMotion2

This is known as "The Bresenham Circle Algorithm". It is a famous algorithm in computer graphics but again in terms of sending as motion commands to actual robot, this is also pretty discontinuous.



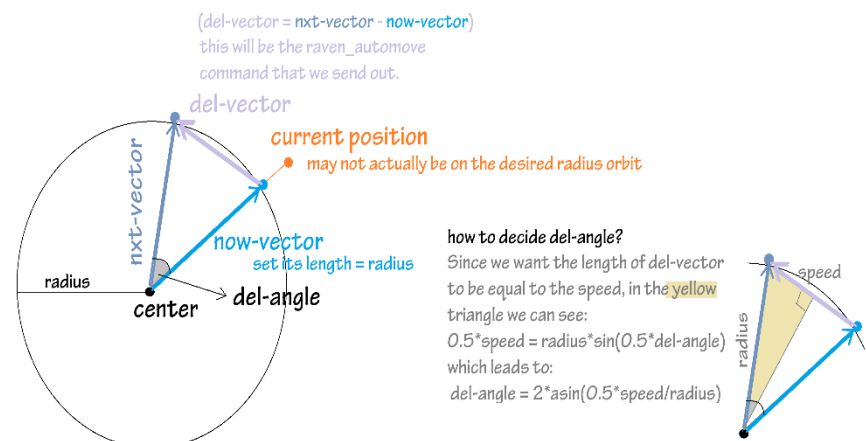
AutoCircleMotion3

This is better at generating customized commands to regulate the distance from center to be around the desired radius. But causes discontinuous commands which makes the robot a little shaky (especially with faster speed).

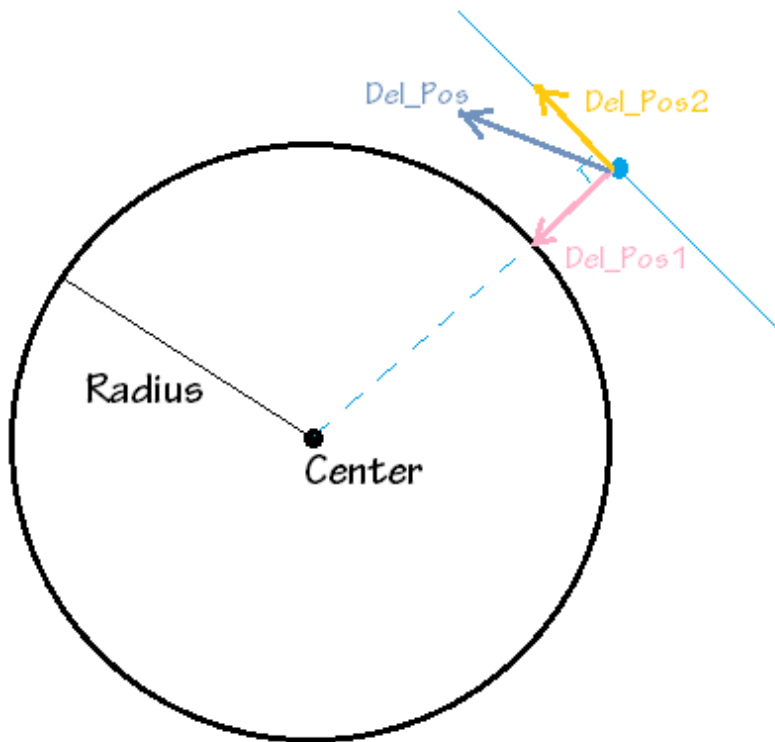


AutoCircleMotion4 (currently in use!!)

This algorithm ensures raven-automove commands to be smooth and continuous. But it does not give any modification when the radius is off. So, we need another mode to tune the radius if the distance from center is too different from desired radius.



4. Regulating Motion Around Desired Radius



$$K_p = 0.000001 * (\text{Speed})^{0.7}$$

$$\text{Error} = \text{abs}(\text{Distance} - \text{Radius})$$

$$K = K_p * \text{Error}$$

$$\text{Del_Pos} = (K) * \text{Del_Pos1} + (1-K) * \text{Del_Pos2}$$

5. Demo Screenshots of AutoCircle generater

Auto Circle Generator Selection Menu:

```

'1' : Increase Circle Radius
'2' : Decrease Circle Radius
'3' : Increase Raven Moving Speed
'4' : Decrease Raven Moving Speed
'5' : Toggle circling direction
'6' : Toggle pause/resume
'7' : Toggle console messages
'8' : Quit

```

```

pos_d[0] = 1061 pos_d[1] = 1061 pos_d[2] = 1061
pos_d[0] = 1061 pos_d[1] = 1061 pos_d[2] = 1061

RADIUS = 1
SPEED = 1

[ INFO] [1472765358.583097032]: talkerAutoCircle publish: raven_automove[98]
[ INFO] [1472765358.625165353]: talkerAutoCircle subscribe: raven_state[1040]
pos[0] = 1071 pos[1] = 1071 pos[2] = 1071
pos[3] = 1071 pos[4] = 1071 pos[5] = 1071
pos_d[0] = 1071 pos_d[1] = 1071 pos_d[2] = 1071
pos_d[0] = 1071 pos_d[1] = 1071 pos_d[2] = 1071

RADIUS = 1
SPEED = 1

[ INFO] [1472765359.582715020]: talkerAutoCircle publish: raven_automove[99]
[ INFO] [1472765359.623574904]: talkerAutoCircle subscribe: raven_state[1050]
pos[0] = 1081 pos[1] = 1081 pos[2] = 1081
pos[3] = 1081 pos[4] = 1081 pos[5] = 1081
pos_d[0] = 1081 pos_d[1] = 1081 pos_d[2] = 1081
pos_d[0] = 1081 pos_d[1] = 1081 pos_d[2] = 1081

RADIUS = 1

```

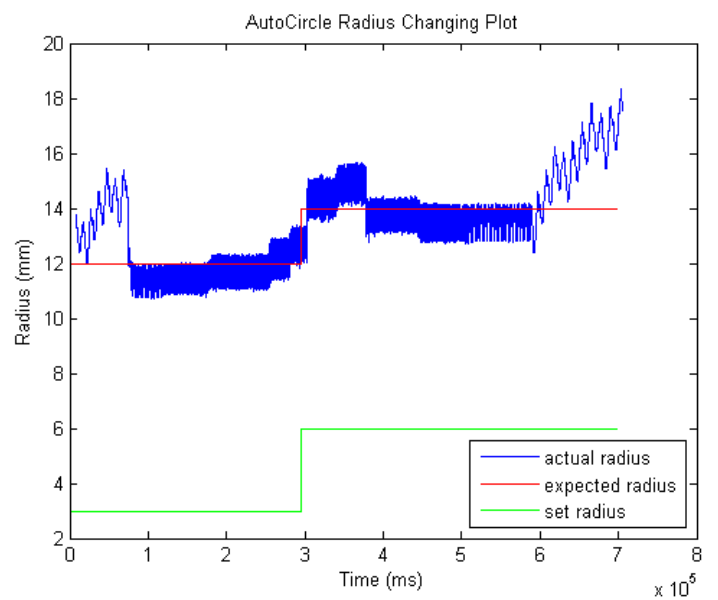
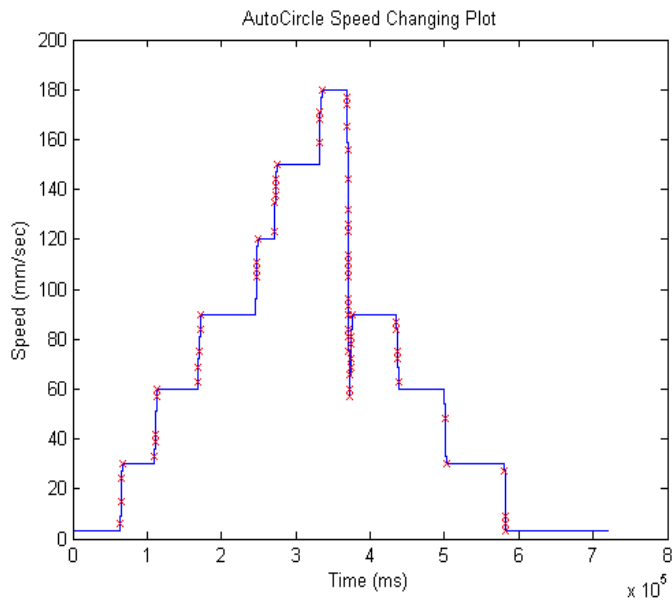
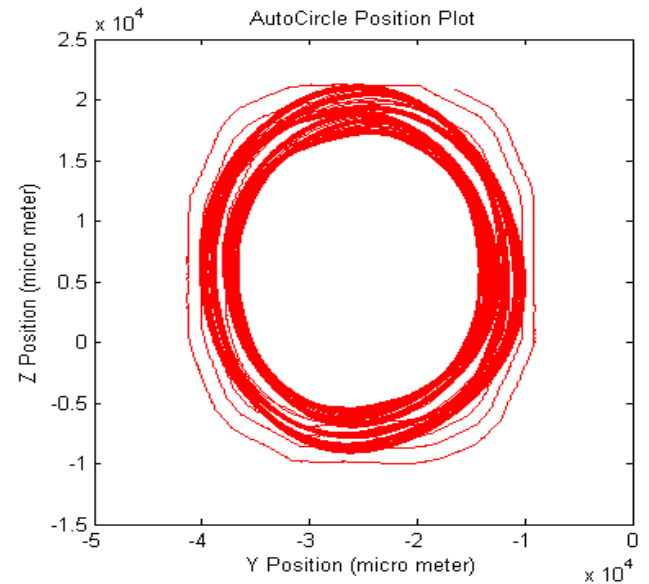
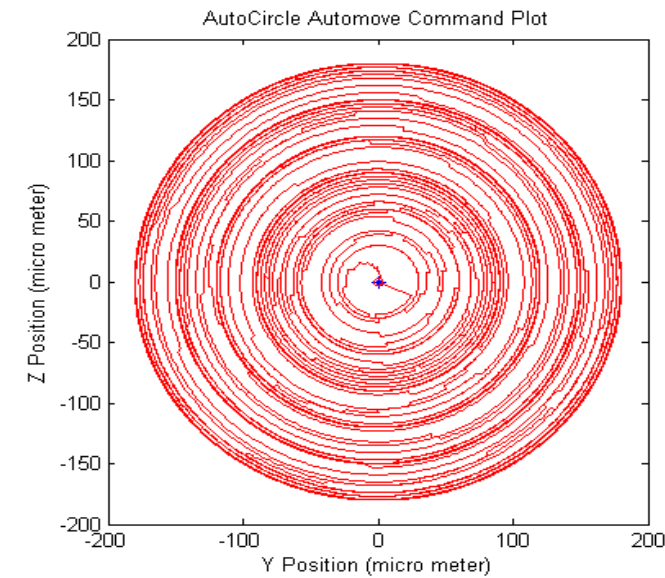
```

[ INFO] [1472765358.622699015]: listenerAutoCircle publish: raven_state[1071]
[ INFO] [1472765359.582907637]: listenerAutoCircle subscribe: raven_automove[98]
data1.xd[0] = (99,99,99)
data1.rd[0] =
0.997007 0.0625567 -0.0454212
-0.0611202 0.997606 0.0323557
0.0473365 -0.0294827 0.998444
data1.xd[1] = (99,99,99)
data1.rd[1] =
-0.638107 0.069689 0.766787
0.716603 -0.310486 0.624563
0.281602 0.94802 0.148184

[ INFO] [1472765359.623292417]: listenerAutoCircle publish: raven_state[1081]
[ INFO] [1472765360.583542681]: listenerAutoCircle subscribe: raven_automove[99]
data1.xd[0] = (100,100,100)
data1.rd[0] =
0.72746 0.607357 -0.319248
-0.476538 0.781968 0.401793
0.493674 -0.140155 0.858279
data1.xd[1] = (100,100,100)

```

6. Data Analysis on Matlab



Radius Level	Speed Level	Actual Radius (distance from center)	K Value
Level 1 (=0.3cm)	Level 1 (=0.3cm/sec)	1.2 cm ~ 1.3 cm	0.02
Level 1 (=0.3cm)	Level 10 (=3cm/sec)	1.2 cm ~ 1.3 cm	0.08 ~ 0.09
Level 1 (=0.3cm)	Level 20 (=6cm/sec)	1.1 cm ~ 1.2 cm	0.15
Level 1 (=0.3cm)	Level 30 (=9cm/sec)	1.1 cm ~ 1.2 cm	0.20
Level 1 (=0.3cm)	Level 40 (=12cm/sec)	1.2 cm ~ 1.3 cm	0.26
Level 1 (=0.3cm)	Level 50 (=15cm/sec)	1.2 cm ~ 1.3 cm	0.3
Level 2 (=0.6cm)	Level 1 (=0.3cm/sec)	1.43 cm	0.018
Level 2 (=0.6cm)	Level 10 (=3cm/sec)	1.35 cm	0.08
Level 2 (=0.6cm)	Level 20 (=6cm/sec)	1.33 cm	0.13
Level 2 (=0.6cm)	Level 30 (=9cm/sec)	1.39 cm ~ 1.40 cm	0.18
Level 2 (=0.6cm)	Level 50 (=15cm/sec)	1.443 cm	0.28