Steps taken for completion of project

1. Setting up Spark V
2. XBee interfacing with Spark V
3. Go-to-Goal and PID controller
4. Scaling up for multiple robots
5. Collision avoidance

1.1 Setting up Spark V

Refer the hardware and software manuals for using the Spark V robot.

1.1.1 Programming the Spark V using AVRDude

1. Copy the AVRDude folder onto your computer
2. Open the Command Prompt in the AVRDude folder
3. Run - stkrun.bat "SparkV.hex" to program the Spark V with the file "SparkV.hex". Similarly you can program any file by giving the location of the file along with the file name.

1.1.2 Programming the Spark V using Atmel studio and AVRDude

1. Copy the AVRDude folder onto your computer
2. Open the `Tools>External` tools in Atmel Studio
3. Name - STK500v2
4. Set the location of AVRDude.exe in the commands box
   For eg `C:\Users\Chirag\Desktop\eYSIP_2017\AVRDude\avrdude.exe`
5. Arguments box - -c stk500v2 -p m16 -P NEX-USB-ISP -U flash:w:`$(TargetName).hex:i`
6. Initial Directory - `$(TargetDir)`
7.

1.2 XBee interfacing with Spark V

The data is received character by character over UART. When a character is received it is very briefly stored in a buffer and a flag is set. This is done internally by the microcontroller. This flag then triggers an interrupt.
The interrupt then decides what to do with this character of data. If it is a valid byte of data it will be stored in the array "data_string_var". After a complete packet of data is received it is copied to the array "data_string".

<#.....#> is a proper string which then saved to "data_string" by the ISR.

Listing 1.1: Interrupt Service Routine

```
ISR(USART_RXC_vect)
{
   previous_data = data;
   data = UDR;

   strcpy((char*) data_string, (const char*)data_string_var);
       //Entire string received!! Save it!!

   if (previous_data==0x3C && data == 0x23)//< and #
   {
      append_on = 1;
      i=0;
   }

   else if (previous_data==0x23 && data==0x3E)//# and >
   {
      append_on=0;
      strcpy((char*)data_string, (const char*)
          data_string_var); //Entire string received!! Save
          it!!
   }

   else if (append_on==1 && data != 0x23)
   {
      data_string_var[i]=data;
      i++;
   }
}
```

1.3 Parsing the String

Listing 1.2: String Parsing Function

```
void update_values()
{
   char parts[10][5];
   char data_string1[40];
   strcpy((char*)data_string1, (const char*)data_string);

   char *p_start, *p_end;
   unsigned char i=0;
   p_start = data_string1;

   while(1)
   {
      p_end = strchr(p_start, '/');
      if (p_end)
      {
         strncpy(parts[i], p_start, p_end-p_start);
         parts[i][p_end-p_start] = 0;
         i++;
         p_start = p_end + 1;
      }
      else
      break;
   }

      x_current = atoi(parts[1]);
      y_current = atoi(parts[2]);
      theta_current = abs(atoi(parts[3])-360+180-360);
          //(0)-(360)
      x_req = atoi(parts[4]);
      y_req = atoi(parts[5]);
      theta_req = abs(atoi(parts[6])-180-360); //(0)-(360)
      trigger = atoi(parts[7]);
      trigger_angle = atoi(parts[8]);
}
```

The string is received in the form

<#id/x_current/y_current/theta_current/x_required/y_required/theta_required/trigger/trigger_angle#>

This is then parsed to obtain the values in int data type.

1.4 Go-to-Goal and PID

Listing 1.3: PID controller

```
int v_left,v_right,R=3.5,L=11.5,V;
float kp, ki, kd;
float w;

kp=.7;
ki=0;
kd=70;
V=400;

distance=sqrt(square(y_req-y_current)+square(x_req-x_current));
destination=(distance<100)?1:0;

if (!x_req || !y_req)
{
   hard_stop();
   destination = 1;
   return;
}

V=(distance<100)?200:400;
kp=(distance<100)?.4:.7;

if(distance>20)
{
   er=theta_current-theta_req; //-360 to 360
   er = atan2(sin(er*3.14/180),
       cos(er*3.14/180))*(180/3.14); //-180 to 180

   integral= integral+er*dt;
   derivative = (er-previous_er)/dt;
   previous_er = er;

   w=kp*er + ki*integral + kd*derivative;
   v_left=((2*V+w*L)/(2*R));
   v_right=((2*V-w*L)/(2*R));

   velocity(v_left,v_right);
   destination=(distance<100)?1:0;
}
```

```
else
{
   hard_stop();
}
```

A Go-To-Goal PID controller is implemented on the microcontroller.
The velocity of the motors is considered to be proportional to the PWM input to the wheels. The PID controller takes care of any non-linearity between the velocity and the PWM input.
The velocity and the kp values are reduced as the robot nears its goal point to avoid overshoot and to avoid oscillations.
Obstacle avoidance behavior is also turned off (by setting destination=1) as the robot nears its goal point so that it does not miss its goal point if another robot is near the goal point.

We use a modified function for velocity. Negative velocity will make the wheel rotate in the opposite direction. Giving a value greater then 255 will have no effect and it will be capped to 255.

Listing 1.4: velocity function

```
void velocity(int left_motor, int right_motor)
{
   if (left_motor>=0 && right_motor>=0)
   {
      forward();
   }

   else if (left_motor<0 && right_motor>0)
   {
      left();
      left_motor=abs(left_motor)+40; //some offset is added so
          that the wheel still turns at lower values
   }

   else if (left_motor>0 && right_motor<0)
   {
      right();
      right_motor=abs(right_motor)+40;//some offset is added
          so that the wheel still turns at lower values
   }
   else
   {
      back();
      left_motor=abs(left_motor);
      right_motor=abs(right_motor);
   }
   OCR1AL = left_motor;   // duty cycle 'ON' period of PWM out
       for Left motor
   OCR1BL = right_motor;   // duty cycle 'ON' period of PWM
       out for Right motor
}
```

```
v_left=((2*V+w*L)/(2*R));
v_right=((2*V-w*L)/(2*R));
```

These equations convert the unicycle model to the differential drive model. The unicycle model gives the state of the robot in terms of its liner velocity 'V' (cm/sec) and its angular velocity 'w'.
The differential drive model gives the state of the robot in terms of the angular velocity (RPM) of each wheels.
While this equations are unnecessary in this case they could be made applicable if we had precise control over the angular velocity of the wheels(RPM).

1.5 Collision avoidance

Listing 1.5: Collision avoidance function

```
void avoid_obstacle()
{
   if (trigger==1)
   {
      hard_stop();
      _delay_ms(100);
      velocity2(90, 90);
      if (trigger_angle > 50) back_mm(50);
      left_degrees(trigger_angle);
      forward_mm(50);
      hard_stop();
   }
   else if (trigger==2)
   {
      hard_stop();
      _delay_ms(100);
      velocity2(90, 90);
      if (trigger_angle > 50) back_mm(50);
      right_degrees(trigger_angle);
      forward_mm(50);
      hard_stop();
   }
}
```

If a trigger is received the robot executes this function. The robot reduces its speed, turns a specified number of degrees and moves forward. If the angle is greater then a certain value(50) the robot moves backwards before taking a turn. This is to avoid a head-on collision with another robot.

1.6 Scaling up