Pavan Kumar Kavvuri Student Number : 1976597

Answer: 1

A Robotic system is be defined as a machine/mechanism that provides intelligent services and information by interacting with their environment via the use of various sensors, actuators and human interfaces. In the current subject, a differential drive robot is used to steer along a line/curve and localise itself from the starting position(hereafter called home) and drive back home on certain conditions. This robot operates indoors with two drive wheels mounted on a common axis, each of which is attached to its own motor and a third wheel is placed in the rear to passively roll along while preventing the robot from falling over. While we can vary the velocity of each wheel, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC - Instantaneous Center of Curvature.

Motion of the robot is governed by a microcontroller/microprocessor that uses sensors and actuators to perceive the environment and generate the control action. In this context, a Romi32U4 is used as a microcontroller, sensors are Pololu QTR-3A Reflectance and 2 brushed DC motors are used as actuators. Encoders make the key element of this control system with its closed loop feedback signals.

This assessment can be divided into two main tasks: (1) Driving from the start, following the line and reach the terminus, and (2) Return home by traversing back the distance from home.

Identifying the black/white space, calibrating pose from the home position, maintaining the set point and checking the confidence in line break are the tasks that run in loop while the robot performs task 1. Accomplishing the task 1, in task 2 robot turns towards home, finalizes its distance travelled from home and resets its encoders to travel back the same distance. The following flowchart shows this decomposition.

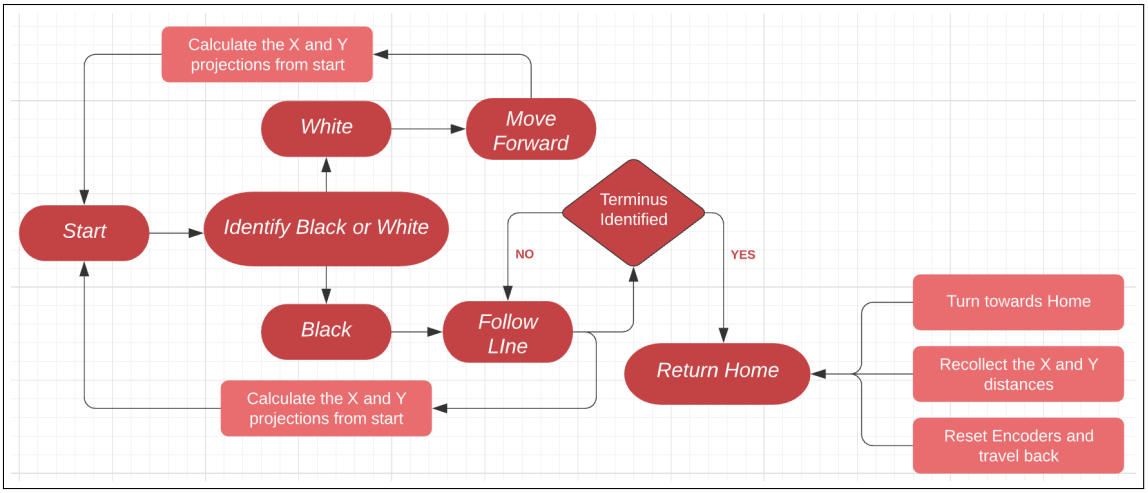

Figure 1: Flowchart of the approach

This decomposition is necessary for the following reasons: (1) as it requires to have distinction between black and white spaces to follow and line, (2) localising itself at every instant is necessary to know how far the robot has travelled from home, (3)Identifying the global maxima (i.e, Terminus) avoiding the local maxima is important to start travelling back to home. The next section discusses in brief about the implementations of these tasks.

Calibrating the set-point prior to the experiment is crucial before we encounter a black line and follow it. Weighted mean of the sensor values is considered as a parameter to define the pose of the robot. Setpoint is defined as a value with high frequency obtained from weighted mean when the middle sensor of the sensor array is on black line and the other two are on white. Deciding the setpoint, we can distinguish black and white spaces. Any deviation from this set point is termed as an error signal and is used by the PID control scheme to regain the state desired. The robot is instructed to steer forward on the white space until a black line is encountered. When it encounters a black line, PID control acts upon it to turn it towards the

black line by changing the direction of rotation of the wheels. Altogether, this results in a smooth motion following the black line. Using the forward kinematics of differential drive robot, orientation, horizontal and vertical distances traversed from the start point are recorded in every instance the robot moves. When the robot reaches the terminus, it is programmed to turn towards the home and traverse the previously recorded horizontal and vertical distances to finally reach home.

```
//Distance between wheels  = 14 cm;
//Radius of wheels (r) = 3.5 cm;
//Encoder count(per wheel) for 1 revolution(2*pi*r)=1425
//Distance per count = 2*pi*r/1425 = 0.015 cm.
//Distance traversed by Left and Right wheels are:
LeftencoderChange = encoderR_count-encoderR_countPrev
RightencoderChange= encoderL_count-encoderL_countPrev
float Right_distance = distancePerCount*LeftencodeChange;
float Left_distance = distancePerCount*RightencoderChange;
encoderR_countPrev = encoderR_count;
encoderL_countPrev = encoderL_count;
// Change in orientation
theta += (Right_distance - Left_distance) / wheelDistance;
meanDistance = (Right_distance + Left_distance)/2;
// Horizontal distance from home
xProj = xProj + meanDistance*cos(theta);
// Vertical distance from home
yProj = yProj + meanDistance*sin(theta);
//Instantaneous center of curvature
```

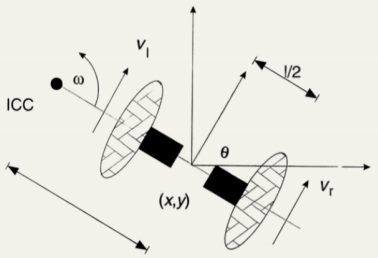

Figure 2: Differential Drive Kinematics pseudo code

Three main elements that posed challenging in this task are PID tuning, understanding differential drive forward kinematics, and turning to desired angle while homecoming. Although challenging, homecoming is the most successful part. For the optimal performance, tuning the parameters Kp, Ki and Kd is necessary. I made the following observations while tuning:

PID Tuning: Initially all the gains were set to zero so that to see the performance without PID. This motion was jerky. Then increased Kp as a floating point value to 0.07 that reduced the sudden rise in accelerations. Then started increased the Kd and Kp together. This step produced smooth motion at the sharp turning although the it's velocity is very low. Then iteratively tuning Ki to 0.0002 produced the desired motion of the robot.

Homecoming: After sensing end of the line, the robot needs to calculate and turn to the angle $180 - \theta$(theta is it's current orientation). This posed a challenge but it was solved simply with an if-else conditional statements. Before turning we had count the total horizontal and vertical distances. Then again travel that distances back finally to reach home. But the main challenge was to reset the encoders and start afresh count. With the help of the library *Romi32U4* we could achieve that. But still the robot couldn't reach the exact start location as there was error in the robot wheel motion resulting deviation from

Uncertainty in the line end: Romi is tasked to return home when it confirms the line break/end with high level of confidence. This confidence can be calculated as probability of detecting line break given all the previous measurements.

$$P(Z_t being white)|Z_{t-1}, Z_{t-2}, Z_{t-3}....Z_1)$$

where Z_t is the measurement at time t. The above expression represents the frequency of encountering the white space given all the previous measurements. High frequency implies high confidence in the line-break. Hence the robot decides that it has reached its destination.

Finally, while disparities in the sensors are inevitable, I would nominate the idea of leveraging the on-board inertial measurement unit(IMU) sensors and fuse them(also called sensor fusion) with encoders information for better estimating the robot's orientation. And allowing the robot to dynamically set the threshold value, adapting to varied illuminance would be a good practice. Mastering the PID tuning techniques with various available methods can also help. Measuring and correcting the error between two wheels helps accomplishing the home coming task.

The implementation of the main tasks on the actual map can be found in fig:3.

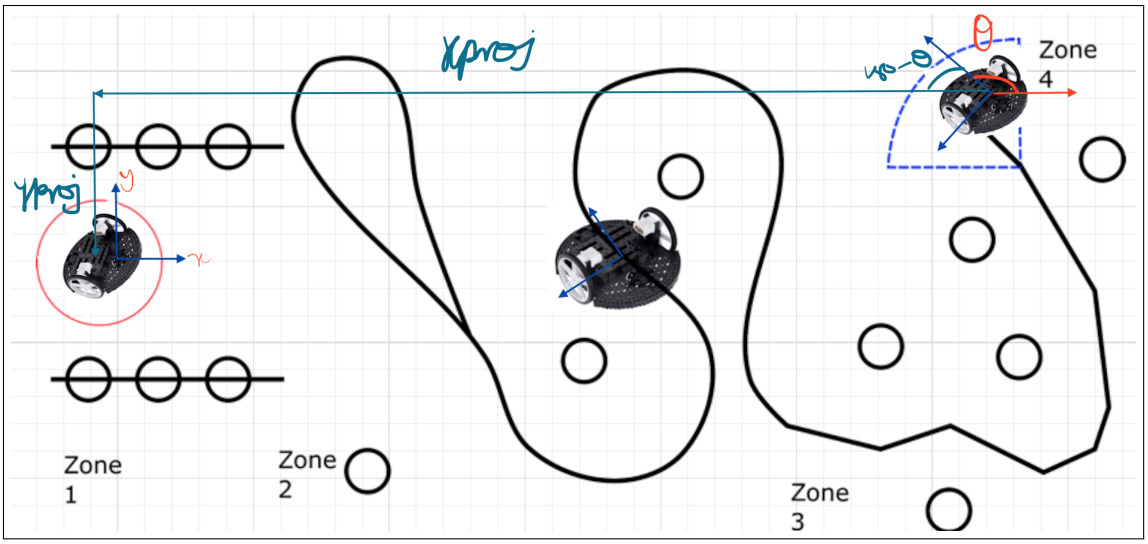

Figure 3: Implementation chart

Answer: 2

The choice of the control algorithm for line-following task .....tricky... 3 popular schemes can be implemented for accurately following a line:

- PID control
- Fuzzy control
- Bang Bang control

With time and complexity in consideration, PID control and Bang Bang control appeared feasible although PID control produces a smooth desired motion along the line. PID control scheme in described here in detail in the context of line following task while the other two are briefly discussed.

PID control: Proportional-Integral-Derivative (PID) control is the most common control scheme used in industry and has been universally accepted in industrial control. Each of these proportional, integral and derivative terms describe how the error signal(deviation from desired outcome) is treated prior to being summed and sent into the plant. In the proportional path, the error term is multiplied by a constant term Kp, in integral path the error is multiplied by Ki and then integrate it and in derivative path by Kd and then differentiate it. These three paths are summed together to produce the controller outputs. The three constant terms are called gains and are adjusted to a particular plant with a defined set of requirements. By tuning these constants, sensitivity of the system to each of these paths is affected.

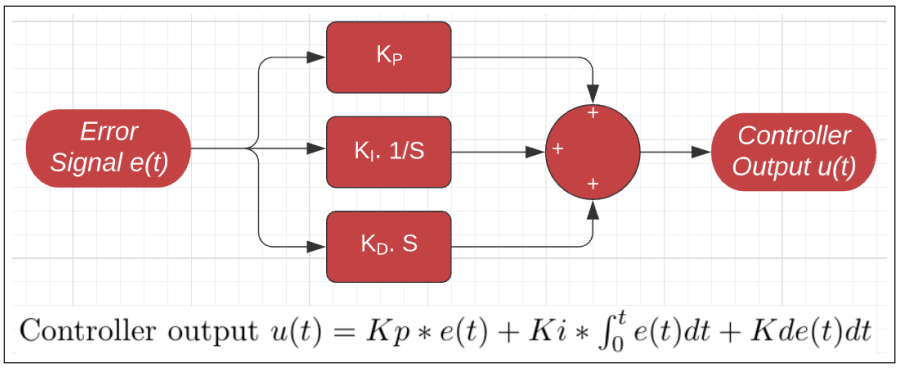

Figure 4: Operation of PID control

PID for Line Following: PID algorithm is used for achieving a desired state of the system In any observable systems, by iteratively acting upon the error signal with its tuned constants, PID algorithm can achieve the desired steady state response in no time. So it is widely accepted algorithm in industry. Although challenging, some established methods like Ziegler-Nichols tuning algorithms can help in tuning to correct parameters.

In the line following task, the desired state is to maintain the set-point constant. While the black space is not always a line but a spline in fact, every following motion creates a deviation from set-point. This deviation is fed back to PID scheme which decides the control action (turn left or right) based on error signal to reduce the deviation towards the set-point. The following code snippet shows how this algorithm calculates the control input and generate the control action.

```
1
2  //Error signal is the difference between weighted mean of 3 sensor values and a pre-defined
       set point.
3  double error = calcWeightedMean() - setPoint; //Error signal is the difference between
       weighted mean of 3 sensor values and a pre-defined set point.
4  double controlInput = calcPidGain(error); // Apply the formula of PID controller gain
5  calcNewTurn(controlInput); // Generate control action based on control input
6
7  int calcPidGain(float currentErr){
8
9      proportional = currentErr;
10     errorIntegral =  currentErr + errorIntegral;
11     errorDerivative =  currentErr - lastError;
12     lastError = currentErr;
13
14     return int(proportional * Kp + errorIntegral * Ki + errorDerivative * Kd);
15 }
16 void calcNewTurn(double inputVal){
17     if(inputVal < 0){
18         moveLeft();
19         analogWrite( L_PWM_PIN, minSpeed - inputVal);
20         analogWrite( R_PWM_PIN, minSpeed - inputVal);
21       }else if(inputVal > 0){
22           moveRight();
23           analogWrite( L_PWM_PIN, minSpeed + inputVal);
24           analogWrite( R_PWM_PIN, minSpeed + inputVal);
25         }
26         else{
27           moveForward();
28           analogWrite( L_PWM_PIN, minSpeed);
29           analogWrite( R_PWM_PIN, minSpeed);
30         }
31  }
```

Listing 1: PID control scheme code

Advantages:

- It only acts on the error signal ignoring all extra measurements of the internal states, reducing measurements, sensors, signal conditioning, maintenance and cost.
- The tuning can be done through trial and error or a look-up table. Not much expertise is needed for tuning.
- Easy to implement in hardware (through filters) or through microcontrollers.
- Efficient and robust against some common uncertainties and unmeasured disturbances

Disadvantages:

- Not really suited for non-linear plants. Performance is low handling strong non-linearities.
- Tuning is challenging with increase in the order of the plant.

Bang-Bang Control: In control theory, a bang–bang controller, also known as a hysteresis controller, is a feedback controller that switches abruptly between two states. If the measured value is above the set-point, execute something else execute some other thing. The output from an individual reflectance sensor will be a voltage between 0 and Vin . This will be read by the Analog to Digital Converter (ADC) in the Romi's Atmel 32u4 and converted to an integer between 0 and 1023. We must convert these numbers into something which is meaningful to us. The simplest way to do this is to convert the output to a binary signal by applying a threshold. The control logic is as follows:

- Line under sensor 1 - Turn Right
- Line under sensor 2 - Go Straight
- Line under sensor 3 - Turn left

```
int threshold = 600;
 if(middleSensor > threshold){
     moveForward();
  }else if(rightSensor > threshold){
        moveLeft();
    }else if(leftSensor > threshold){
        moveRight();
    }else{
        Stop();
      }
```

Listing 2: Bang Bang control scheme code

Table 1: PID over Bang-Bang Controller

#	Bang-Bang	PID
1	Simpler to run and code	Involves calculus math to generate control input
2	Faster spin up times but inaccurate in range	Far more accurate
3	Not ideal for sharp turns	Ideal
4	Not highly sensitive to error signal	Control input is highly sensitive to error signal
5	Sudden accelerations can wearout the motor gears	Results in smooth trajectories with no jerks
6	No parameters to tune except for the set-point/threshold	Tuning plays a major role in the optimal performance. It requies to tune Kp, Ki and Kd
6	Ideal for small scale applications	Ideal for industry wide applications

Fuzzy Control: Fuzzy logic control relies on Fuzzy Inference System to control the robot motion. One can choose any number of membership functions depending upon the application. Here, 5 membership functions are ideal and they are: Hard Left, Left, Forward, Right, and Hard Right. Initially, sensor outputs are measured to decide which combination of values best fit the membership functions. A rue-base is developed based on the decisions. Depending upon the sensor values, only few rules get fired, and the area under these membership functions is evaluated to decide the controller output. This is called defuzzification.

Answer: 3

Self-Driving car has always been a fantasy for me until I knew about the *Stanley: The Robot that Won the DARPA Grand Challenge*. I would like to carry out a systematic study on autonomous mobile robots to examine their performance in the most challenging environments like agricultural fields. For eg., let us suppose a robot is tasked to navigate autonomously in a vast agricultural field and count the number of fruits on trees by avoiding collisions with obstacles.

The key elements of such robotic system include LIDAR, RGB-D vision cameras, a GPS antenna and stereo cameras. GPS antennas provide the data on position, pitch and heading. A rotating LIDAR bounces laser beams to create a 3D map of the terrain. RGB-D camera gives us color of the fruits and depth of the obstacles. Stereo cameras determine the position and orientation of a robot by analyzing the associated camera images which is called Visual Odometry. I would decompose this challenge in the following way:

- **Sensor interfacing**: To receive and timestamp data from multiple sensors.
- **Perception of Sensor data**: Map data to unscented Kalman filter(UKF) to estimate robot's state, pose, and velocities.
- **Laser Terrain Mapping**: To safely avoid obstacles, robot must be capable of accurately detecting non-drivable terrain at a sufficient range to stop or take the appropriate evasive action. Lasers are used as the basis for Stanley's short and medium range ob
- **Path Planning**: Responsible for regulating the steering, throttle, and brake response of the robot and set the trajectory of the vehicle in steering and velocity space.
- **Sensing and Counting**: Identify the trees, localise the fruits on it and count them using RGB-D camera. Images are processed using convolutional neural network.

Robot state estimation is a key prerequisite for precision driving across the agricultural terrains. Inaccurate estimations can cause the robot to drive against the obstacles, or build terrain maps that do not reflect the state of the robot's environment, leading to poor driving decisions. With programs in on-board computers, the robot should decide how to navigate through the mapped terrain and unmapped obstacles in real time.

To localise itself in the mapped terrain, the robot fuses data from GPS receivers, Visual odometry sensors and Inertial Measurement Units (IMU). This process of combining data from multitude of sensors(also called sensor fusion) reduces the uncertainty in measuring it's location. An Unscented Kalman Filter (UKF) incorporates observations from these sensors to estimate the state(position, velocity etc,) of the robot.

My approach to the above challenge would be influenced by the following impediments in Romi that I encountered while developing and executing the line-following task.

- For processing camera images, Romi needs to be backed up with additional processing power from alternate processors like Raspberry Pie.
- Depending merely on odometry for navigation tasks wouldn't suffice for unstructured environments like agriculture. This is a key learning from this Romi assessment.
- Operating in a varied illuminance is crucial for my challenge under consideration, so I would incorporate thermal and IR cameras to avoid illuminance problems.

The performance of this model can be evaluated as follows:

For example consider an agricultural field lane with poles as obstacles and trees spread on either side of the lane. Now a robot is tasked to detect and count the fruits, while driving through the navigable terrain by avoiding obstacles and mapping the environment in 3D. At the end the lane it should give the correct count of the fruits. The robot must have robust path planning from the observations of multiple sensors. Besides, a sensor outage should not effect it's performance in a great deal.