모바일 로봇 프로그래밍

결과 보고서

Mobile Robot Programming

Spring 2019

정 설 영

2조

컴퓨터학부 이병도

컴퓨터학부 홍두표

컴퓨터학부 김승재

경제통상학부 김만기

**목 차**

1. 개요
2. 진행 과정
3. 구현 방법
4. 참조 문헌
5. **개요**

저희 조는 Turtlebot을 이용해 주차장 내에서 주차 표지판을 찾아서 자율주차를 수행하는 것을 가정하고 프로젝트를 진행하였습니다.

주어진 주차장에서 Turtlebot은 자율 주행을 실시합니다. 실제 자동차라고 가정했을 때 우선적으로 장애물에 부딪히지 않는 것이 최우선이라고 생각해서 일정 속도로 주차장 내부를 주행하며 장애물을 만날 시 Turtlebot은 장애물이 없는 방향으로 다시 자율 주행을 계속 합니다. 자율주행을 하며 주차공간 표지판을 발견하면 그 공간에 주차를 하게 됩니다.

1. **진행 과정**

|  |  |
| --- | --- |
| 1주차 | * 브레인스토밍을 통한 아이디어 생산 * 회의를 통한 프로젝트 주제 결정 * 프로젝트 주제에 관련된 자료조사 |

|  |  |
| --- | --- |
| 2주차 | * 구현에 필요한 기능 분담 * 주차 공간 탐색 기능 * 장애물 회피 기능 구현 * 주차공간 표지판 감지 기능 구현 * 주차 기능 구현 |

|  |  |
| --- | --- |
| 3주차 | * 기능별 시스템 통합 * 테스팅 * 버그 수정 |

1. **구현 방법**

3.1 System Layout

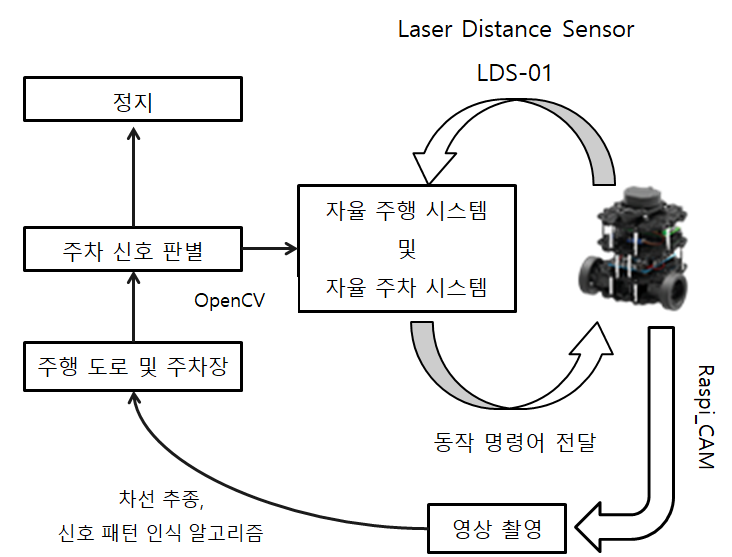


그림 1. Flow chart

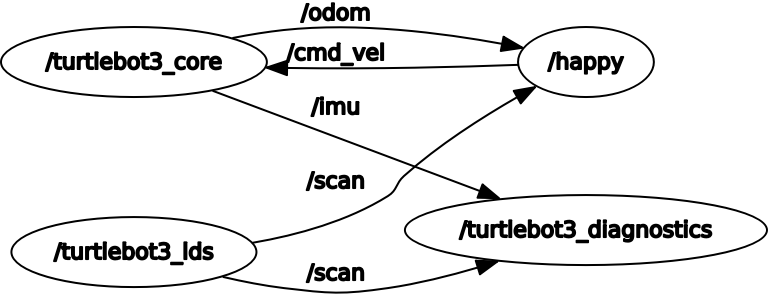
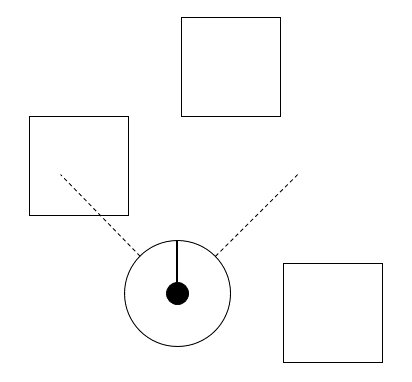
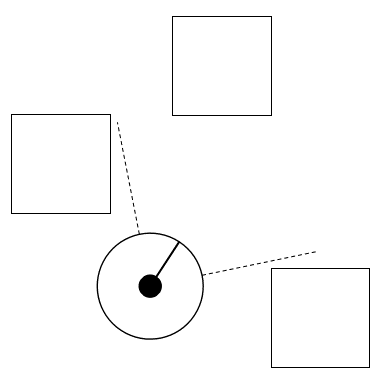


그림 2. rqt\_graph

3.2 장애물 회피 알고리즘

<그림 1> <그림 2>

로봇은 주행 중에 계속해서 자신의 Laser scan 데이터를 수집합니다. 장애물 회피 알고리즘에서 로봇이 특정 물체와 어느 정도만큼 가까워지면 장애물로 인식해야 되는지 그 거리를 설정하였습니다. 로봇이 주행을 하다 Laser scan 데이터를 통해 파악된 물체와 자신이 설정된 거리보다 가깝다고 판단되면, 장애물로 인식을 하고 (그림 1),(그림 2) 자신의 주행 방향을 기준으로 장애물이 위치한 쪽과 반대되는 방향으로 방향을 틀어 주행하게 하였습니다.

bool isnan(T value)

{

return value != value;

}

double average(vector<Vec3d> &laserScanXY, Point cur)

{

double ret = 0;

for(int i = 0; i < laserScanXY.size(); i++)

{

double xdiff = laserScanXY[i][0] - cur.x;

double ydiff = laserScanXY[i][1] - cur.y;

ret += sqrt(xdiff \* xdiff + ydiff \* ydiff);

}

if(laserScanXY.size() == 0) return INT\_MAX;

return ret / laserScanXY.size();

}

double avg = average(laserScanXY, Point(odom.pose.pose.position.x, odom.pose.pose.position.y));

if(isnan(avg)) continue;

if(avg > delta) continue;

다음과 같은 과정을 통해서 스캔으로 받아들일 수 있는 거리 내부에 장애물이 있을 경우 회전을 하게 됩니다.

회전의 경우에는 장애물이 없을 때 까지 계속해서 while문을 통하여 실행이 됩니다. 핵심부분은 다음과도 같습니다.

while(ros::ok() && !bDone) {

ros::spinOnce();

tf::Transform currentTransformation = getCurrentTransformation();

tf::Transform relativeTransformation = initialTransformation.inverse() \* currentTransformation ;

tf::Quaternion rotationQuat = relativeTransformation.getRotation();

double dAngleTurned = rotationQuat.getAngle();

if( fabs(dAngleTurned) > fabs(dRotation))

{

bDone = true;

break;

} else {

pubTeleop.publish(baseCmd);

loopRate.sleep();

}

}

3.3 다각형 형태 및 색 인지 알고리즘

로봇의 라즈베리파이 카메라를 통해 들어오는 압축된 이미지를 실시간으로 받아서, 가장 큰 사각형과 삼각형을 검출한다. 우선 그레이이미지로 변환한 뒤, 윤곽 점들을 추출하고, 이를 이어서 다각형과 꼭짓점 개수 및 넓이를 통해 찾고 있는 알맞은 다각형(사각형 및 삼각형)을 인지한다.

이후, 해당 다각형 정보를 통해 이미지에서 적절한 bound 사각형을 만들고 해당 영역을 잘라낸다. 잘려진 영역의 RGB값을 픽셀마다 확인하여, R, G, B 특정 픽셀의 개수를 계산하여 다각형 내부의 색을 R, G, B로 판단한다.

if(approx.size() == 3 && isContourConvex(Mat(approx)) && fabs(contourArea(Mat(approx))) > 1000)

다각형이 삼각형인지 검사하고, 그 크기가 너무 작지 않은지 판단하는 부분

|  |
| --- |
| Rect crop\_img(Mat& img, vector<Point>& largest\_square) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| … |

|  |
| --- |
|  |
|  |

|  |
| --- |
| sort(largest\_square.begin(), largest\_square.end(), [](Point a, Point b) {return a.x < b.x ;} ); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| topLeft = largest\_square[0]; |
|  |

|  |
| --- |
| if(largest\_square[0].y > largest\_square[1].y) |
|  |

|  |
| --- |
| topLeft = largest\_square[1]; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| bottomRight = largest\_square[2]; |
|  |

|  |
| --- |
| if(largest\_square[2].y < largest\_square[3].y) |
|  |

|  |
| --- |
| bottomRight = largest\_square[3]; |
|  |

|  |
| --- |
| Rect ret(topLeft, bottomRight); |
|  |

|  |
| --- |
| return ret; |
|  |

}

실제 추출되는 사각형은 평행사변형이나 사다리꼴일 수 있으므로, 반시계 방향으로 점들의 순서를 정렬한 뒤, 해당하는 사각형 영역을 이미지에서 잘라내는 부분.

|  |
| --- |
| int b = img.at<Vec3b>(i, j)[0]; |
|  |

|  |
| --- |
| int g = img.at<Vec3b>(i, j)[1]; |
|  |

int r = img.at<Vec3b>(i, j)[2];

각 픽셀에 접근하여 RGB 값을 추출하는 부분. OpenCV에서 픽셀 값에 접근할 때, RGB값은 BGR순서로 저장돼 있다는 점을 이용하여 [0]을 blue에, [1]을 green에, [2]를 red에 저장한다.

3.4 주차 알고리즘

다각형 형태 및 색 인지 알고리즘을 통해 주차공간 표지판을 감지하면 주차공간 표지판의 일정거리 앞에 멈추고 종료함으로써 주차를 하게 됩니다.

double angle\_;

double distance\_;

bool happy\_flag;

void scanCallback (const sensor\_msgs::LaserScan::ConstPtr& scan\_msg)

{

if(!happy\_flag) return;

for (unsigned int i=0;i< scan\_msg->ranges.size();i++)

{

std::cout << distance\_ << std::endl;

if (scan\_msg->ranges[i]<distance\_)

{

distance\_ =scan\_msg->ranges[i];

angle\_ = ((double)i\*scan\_msg->angle\_increment)+scan\_msg->angle\_min;

}

}

}

bool parking(ros::Publisher &vel\_pub\_)

{

geometry\_msgs::Twist vel;

ros::Rate loop\_rate(500);

bool done = true;

while(ros::ok() && done)

{

//check gSign

if(gSign == STOP) continue;

if (distance\_ >= 0.7 && distance\_ <= 0.9)

{

if (angle\_ <= 0.0)

{

vel.linear.x=0.2;

vel.angular.z=0.6;

vel\_pub\_.publish(vel);

}

else

{

if (angle\_>0)

{

vel.linear.x=0.2;

vel.angular.z=-0.6;

vel\_pub\_.publish(vel);

}

}

}

else if(distance\_ <= 0.5) // 장애물 앞에서 멈추는 거리 조절

{

done = false;

vel.linear.x=0.0;

vel\_pub\_.publish(vel);

}

else

{

vel.linear.x=0.15;

vel.angular.z=0.0;

vel\_pub\_.publish(vel);

}

distance\_ = 10;

ros::spinOnce();

loop\_rate.sleep();

}

return done;

}

1. **참조 문헌**

참조 사이트

* [www.ros.org](http://www.ros.org)
* <http://emanual.robotis.com/docs/en/platform/turtlebot3/applications/#automatic-parking>

참조 오픈소스

<https://github.com/ROBOTISGIT/turtlebot3_applications/tree/master/turtlebot3_automatic_parking>

- <https://github.com/tysik/obstacle_detector>