VISUAL BASED GUIDANCE FOR AUTONOMOUS VEHICLES

INTRODUCTION:

An autonomous car (also known as a driverless car, self-driving car, robotic car) and an unmanned ground vehicle is a vehicle that is capable of sensing its environment and navigating without human input.

Autonomous cars use a variety of techniques to detect their surroundings, such as radar, laser light, GPS, odometry and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage. Autonomous cars must have control systems that are capable of analysing sensory data to distinguish between different cars on the road.

Here we will be using computer vision to detect the surrounding and any obstacles present in the path. The primary goal will be to implement obstacle avoidance as path planning and autonomous driving would require huge data sets and the timeframe is not enough for that.

Stereo camera (DUO3D) will be the primary source of information. A stereo camera is nothing but a combination of two cameras separated by a distance which basically act like human eyes.

We get into Camera calibration, scene reconstruction, depth maps, etc. to extract information from the obtained real time images.

The processing is done in OpenCV and the corresponding code will be given as an opensource.

A similar project done by Alejandro was taken for reference and I have used some of the libraries and codes as such. A hyperlink to the project is provided in the reference(To be filled later).

PROJECT PARTS

1. Hardware –
   1. A 3D printed structure of the car (chassis with all the mountings).
   2. Wheels.
   3. A thrust/DC motor for movement.
   4. A servo motor to steer.
   5. Battery.
   6. Cooling fan(LANSU).
   7. A 2.4 GHz RF receiver.
   8. Arduino UNO
   9. Linux System
   10. Parts still required

1.10.1- Battery charger

1.10.2- RF Transmitter

1.10.3- Raspberry Pi

1. Software –
   1. DUO3D drivers and application, for calibration, and camera related functions.
   2. OpenCV –
      1. To access the camera and process the obtained images.
      2. Corresponding libraries to access the stereo camera.
   3. Libraries in Arduino IDE to control the movement of the car.
   4. NOTE: Right now, we are using OpenCV in raspberry Pi for all the processing. Clarification required: If RPi can handle the computational cost of the stereo camera. If no, revert to the Linux system, otherwise stick to RPi.

### Camera calibration

Camera calibration is a required step for most computer vision applications using stereo cameras. Its aim is to remove any distortion caused by the lenses and to align both camera planes. Without this step, the algorithms would fail to compute a precise 3D image suitable for path planning.

Making the calibration procedure fast and easy has been one of the most challenging tasks when writing the code for the stereo camera.

Calibrate the camera using the following call (if the provided calibration file does not exist it will be created automatically):

|  |  |
| --- | --- |
| 1  2  3 | string CALIBRATION\_FILE = &amp;quot;&amp;lt;project directory&amp;gt;/data/stereo\_calibration\_parameters.xml&amp;quot;;  string OUTPUT\_FOLDER = &amp;quot;&amp;lt;project directory&amp;gt;/data/&amp;quot;;  stereoCam.calibrate(CALIBRATION\_FILE, OUTPUT\_FOLDER); |

If you don’t want to save the results, just omit OUTPUT\_FOLDER:

|  |  |
| --- | --- |
| 1 | stereoCam.calibrate(CALIBRATION\_FILE); |

Once rectified (see the code at the repository), the stereo images look curved at the margins. A scaling parameter (“alpha” in function stereoRectify) lets you choose whether to crop the image to the valid region of pixels or let it as it is:

|  |  |
| --- | --- |
| 1  2  3  4 | //OpenCV's algorithm  stereoRectify(cameraMatrix0, distCoeffs0, cameraMatrix1, distCoeffs1, imageSize,  RInitial, TInitial, R1, R2, P1, P2, Q, CALIB\_ZERO\_DISPARITY, alpha,  imageSize, validRoi[0], validRoi[1]); |