Autonomous Robot Navigation Using Optical Flow-Based Obstacle Avoidance

Yi-Chung Chen Vamshi Kalavagunta Wei-Li Su Zidong Zhao Ji Liu
University of Maryland - College Park
College Park, MD 20742
ychen921@umd.edu,Vamshik@umd.edu,weisu77@umd.edu,zzhao110@umd.edu,liuji@umd.edu

Abstract

This project focuses on utilizing optical flow information from a single camera to enable autonomous navigation in a complex environment while ensuring obstacle avoidance. Optical flow captures the movement of objects or features in the image resulting from the camera's motion relative to the scene. Drawing inspiration from Souhila and Karim's work, our strategy involves extracting optical flow, focus of expansion, time-to-contact, and depth map data to control the robot's wheel velocity, which is similar to the idea of a Braitenberg vehicle. This approach enables the robot to effectively navigate and evade obstacles. Our project involves implementing this pipeline on a mobile robot platform powered by Raspberry Pi 3 and equipped with a Picamera 2. To assess the algorithm's effectiveness, we will conduct rigorous experiments using our physical robot, including navigation within an enclosed arena with obstacles placed, to evaluate its robustness and performance in real-world scenarios.

1. Introduction

Autonomous mobile robots are machines that can move and perform tasks in an environment without human intervention or guidance. They have many potential applications in various domains, such as exploration, surveillance, rescue, transportation, and entertainment. However, one of the main challenges for autonomous mobile robots is to navigate safely and efficiently in dynamic and unstructured environments, where they may encounter various obstacles and hazards.

To achieve obstacle avoidance, autonomous mobile robots need to perceive their surroundings and plan their actions accordingly. One of the common ways to do this is to use range sensors, such as LiDAR or Sonar, which can measure the distance and direction of obstacles in the robot's vicinity. However, these sensors have some limitations, such as limited range, poor resolution, high cost, and susceptibility to noise and interference. Moreover, they may not be able to detect some types of obstacles, such as transparent or reflective surfaces.

Therefore, alternative methods that use vision-based sensors, such as cameras, have been explored by researchers. Vision-based sensors have several advantages over range sensors, such as a wider field of view, higher resolution, lower cost, and richer information content. By processing the image sequence captured by an embedded camera, a robot can extract useful information about the environment, such as the presence and location of obstacles, the direction and speed of motion, and the time-to-contact (TTC) with obstacles.

One of the vision-based methods for obstacle avoidance is based on optical flow, which refers to the apparent motion of features or objects in the image that is caused by the relative motion between the camera and the scene. Optical flow provides valuable information about the environment, such as the presence and location of obstacles, the direction and speed of motion, and the time-to-contact (TTC) with obstacles. By extracting and processing this information, a robot can adjust its motion accordingly to avoid collisions.

In this project, we implemented an algorithm for autonomous robot navigation using optical flow-based obstacle avoidance. The algorithm is based on the work of Souhila and Karim (2007), who used optical flow to control the wheel torque of a robot such that it avoids obstacles while navigating the environment. We tested this approach by implementing it on a real physical robot equipped with a camera and a Raspberry Pi. We also evaluated the performance of our algorithm through experiments in different scenarios.

The main contributions of this paper are:

1. We implemented an optical flow-based obstacle avoidance algorithm on a real physical robot platform that runs on Raspberry Pi 3 and is equipped with Picamera 2.

2. We tested the robustness of our algorithm in various environments with different obstacles.

3. We analyzed the advantages and limitations of our algorithm.

2. Literature review

In this literature review, we explore the utilization of optical flow in visual navigation systems for obstacle avoidance. Initially, we introduced the concept of optical flow and delve into the various methods employed for its computation. We then investigated how optical flow can be leveraged to extract crucial information such as the focus of expansion (FOE), time to collision (TTC), and depth maps from image sequences. Subsequently, we scrutinize existing algorithms that employ optical flow to govern the motion of robots for effective obstacle avoidance. Lastly, we shed light on the challenges and unresolved issues that persist within this domain.The review is based on the work by Souhila K., Karim A.(2007)[1].

3. Algorithm design

In this section, we describe our algorithm design for autonomous robot navigation using optical flow-based obstacle avoidance. We first present our hardware setup and software framework for implementing our algorithm on a real physical robot. Then we explain our algorithm pipeline in detail, which consists of four steps: computation of optical flow using images captured with camera; extraction of FOE from optical flow; calculation of TTC and depth map; control of wheel velocity based on optical flow information. We had provided the block diagram of the navigation system algorithm below.

4. Methodology

In this project, we first compute the optical flow using captured frames. It can be computed using sparse or dense optical flow methods. Then the Focus of Expansion (FOE) is extracted from the optical flow. This is the projection of the translation vector of the camera on the image plane. Moreover, the Time-to-Contact (TTC) can be calculated easily from FOE. TTC is computed for each pixel in the image, and by multiplying it with the robot's velocity at the moment, one obtains the depth map. Last, the wheel velocity of the robot can be controlled by optical flow.

4.1. Hardware

The robot used in this project is shown in Figure 2. The robot is equipped with a Raspberry Pi 3B as the onboard computer and a Picamera 2 in the front. The robot has 4 motors but the left two wheels receive the same control input and so do the right two wheels. The robot is essentially a differential drive robot.

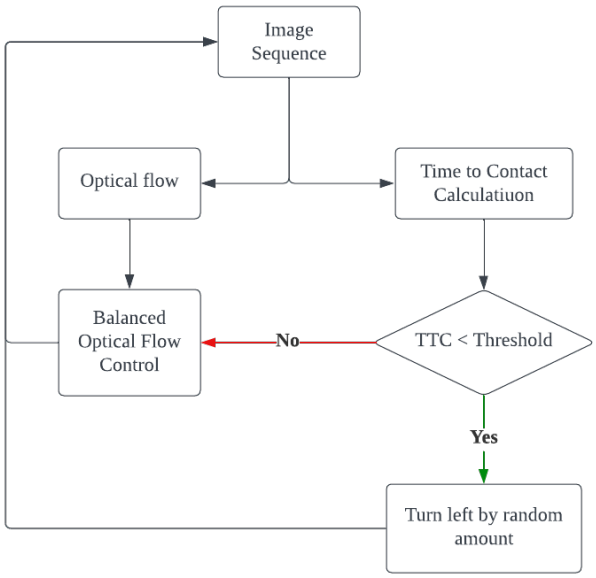


Figure 1. Object avoidance navigation system.

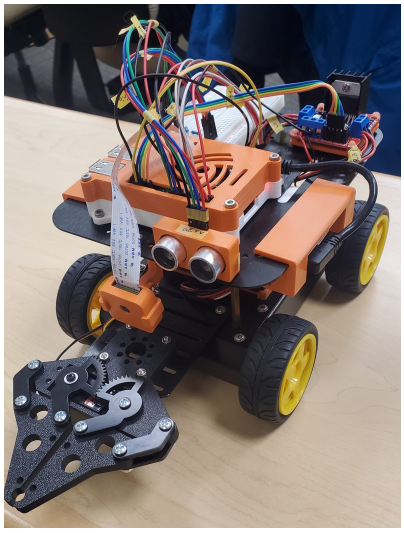


Figure 2. Mobile robot

4.2. Optical Flow

The observed motion between objects and the camera is known as optical flow. Generally speaking, optical flow is caused by the movement of the object itself or the movement of the camera. The Lucas-Kanade method is an algorithm to track points on a video to compute optical flow.

The Lucas-Kanade method is based on the Brightness constancy assumption. The main idea is that the brightness of a certain point in the scene should be roughly the same from one frame to the other if the subsequent frames are

taken within a short amount of time. The simplified Brightness Constraint Equation is shown below, where E_x, E_y are spatial gradients, E_t is the temporal gradient, and u and v are the optical flow, which is the velocity of the physical points projected on the image plane.

$$E_x u + E_y v + E_t = 0 \tag{1}$$

The spatial gradient can be computed using various gradient operators such as Sobel operator, Roberts' cross operator, and Prewitt operator. The temporal gradient is the difference in pixel intensity between two consecutive frames. Therefore, the goal is to compute the optical flow u and v, which are unknown. Solving u and v for each individual pixel is an under-constrained problem. However, we can compute this by assuming that within a small window in the image, the optical flow is similar across the pixels and then resort to the least square approach. Thus, for example, using a 5x5 window gives us 25 equations from which we can solve u and v that minimize the fitting error.

$$Ax = b$$

$$A = \begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{25}) & E_y(p_{25}) \end{bmatrix}, x = \begin{bmatrix} u \\ v \end{bmatrix}, b = \begin{bmatrix} E_t(p_1) \\ E_t(p_2) \\ \vdots \\ E_t(p_{25}) \end{bmatrix}$$

We can solve Ax = b by the least square method where:

$$x = (A^T A)^{-1} A^T b$$

The $A^T A$ is the same matrix as seen in the Harris Corner Detection algorithm:

$$A^T A = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

When the eigenvalues are small, the gradient in all directions is weak, which also means the pixels within the window are likely texture-less and have no good features to track. If one of the eigenvalues is significantly larger than the other, it means that the windowed area contains an edge. Lastly, two large eigenvalues correspond to a corner, which suggests the windowed area is highly textured and is a good point to track.

In this project, we first read two consecutive frames and convert them to grayscale. Next, we calculate the spatial derivatives E_x, E_y and temporal derivatives E_T. Then we set a 15x15 window which can get 225 equations that we can solve $Ax = b$ by least square method and slide the window through the frame. However, in order to find the good feature points to compute optical flow, we only kept the corners whose eigenvalues λ_1, λ_2 are large.

Sparse optical flow methods such as the Lucas-Kanade method output optical flow of good feature points. The

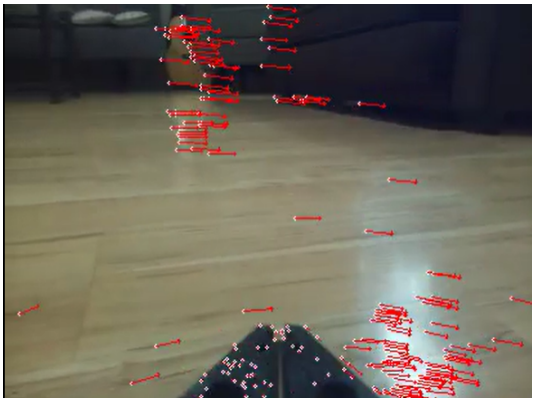


Figure 3. Optical Flow: Lucas-Kanade.

Dense optical flow gives the flow of the entire image. Although Dense optical flow would give us higher accuracy since it gives us the flow of the whole frame, the computation is expensive. Gunnar Farneback method is a way

Figure 4. Optical Flow: Gunnar Farneback.

to compute dense optical flow based on Polynomial Expansion. It first approximates the windows of the image by quadratic polynomials. Then, estimate the displacement fields from coefficients of polynomial expansion. It computes the magnitude and direction of optical flow from the optical flow array and visualizes flow direction by hue, and magnitude using the HSV colorspace.

4.3. Focus of expansion

Focus of Expansion is a point in an image or video sequence that represents the projection of the camera's translational motion onto the image plane. In the presence of translational motion, all optical flow vectors will diverge from this point. The FOE is a key concept used in various computer vision tasks, such as estimating the time-to-contact (TTC), determining the direction of motion, or calculating depth information. By analyzing the FOE, we can gain insights into the dynamics of the scene and use it for

tasks like obstacle avoidance, navigation, or scene understanding.

The FOE can be determined by applying a simple rule to the flow vectors: the horizontal component h_L of a flow vector L on the left side of the FOE is negative, while the horizontal component h_R of a flow vector R on the right side of the FOE is positive, as shown in fig 2. To find the horizontal coordinate of the FOE, we can count the signs of the horizontal components along each column of the image and choose the column that has the smallest difference between negative and positive values. This means that most of the flow vectors diverge horizontally around this column. Similarly, to find the vertical coordinate of the FOE, we can count the signs of the vertical components V_T and V_D along each row of the image and choose the row that has the smallest difference between negative and positive values. This means that most of the flow vectors diverge vertically around this row. The intersection of these columns and rows gives us the FOE coordinates.

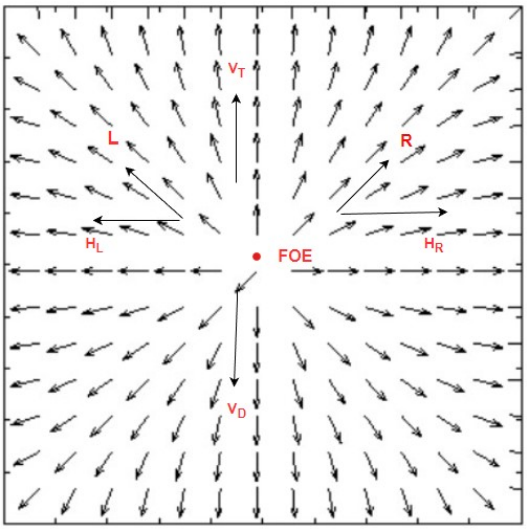


Figure 5. Focus of expansion.

The FOE is computed by finding the horizontal and vertical coordinates that minimize the difference between the number of leftward and rightward flow vectors, and the number of upward and downward flow vectors, respectively.

Let $F(x, y) = (u, v)$ be the optical flow vector at pixel (x, y) in the image. The horizontal coordinate of the FOE, x_{foe}, is given by:

$$x_{foe} = \arg\min_{x} \left| \sum_{y=0}^{h} (F(x,y)_u < 0) - \sum_{y=0}^{h} (F(x,y)_u > 0) \right| \tag{2}$$

where h is the height of the image, $F(x, y)_u$ is the horizontal component of F(x, y).

Similarly, the vertical coordinate of the FOE, y_{foe}, is given by:

$$y_{foe} = \arg\min_{y} \left| \sum_{x=0}^{w} (F(x,y)_v < 0) - \sum_{x=0}^{w} (F(x,y)_v > 0) \right| \tag{3}$$

where w is the width of the image, $F(x, y)_v$ is the vertical component of F(x, y). The FOE is then given by (x_{foe}, y_{foe}).

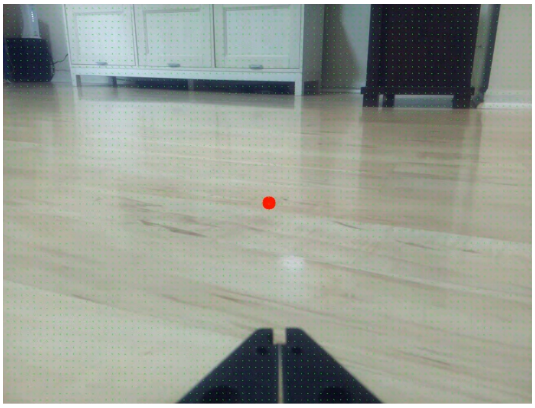


Figure 6. FOE point calculation.

Fig. 3 shows the result of the FOE calculation in our indoor scene, where the FOE is shown by the red circle in the image.

The FOE provides useful information for obstacle avoidance because it indicates where the robot is heading and how far it is from potential collisions. By comparing the location of obstacles with respect to the FOE, we can determine whether they are in front or behind, left or right, and near or far from the robot. Moreover, by tracking the changes in optical flow and FOE over time, we can estimate how fast we are approaching or moving away from obstacles and adjust our speed and direction accordingly.

4.4. Time To Contact Method 1

The Time to contact or Time to collision (TTC) is a method that can predict the duration of time before colliding with objects. TTC is calculated from the distance of the good feature points and the image velocity. The image velocity includes the rotational velocity and the translational velocity. In this project, the testing mobile robot mostly travels in line segments, and thus we are only concerned with the translational velocity. Using the translational optical flow(V_t), the TTC can be calculated as follows:

$$TTC = \frac{\Delta_i}{|V_t|} \tag{4}$$

where Δ_i is the distance of the good feature points (x_i, y_i) from the FOE and the length of the vector lines in the image

indicates the flow speed. The estimation time is a relative rate to show contact objects.

After calculating the TTC of the good feature points, we divide the image into blocks and compute the mean of the TTC in each block. The time to contact in the TTC blocks, which represent good feature points extracted from obstacles, can be estimated. Fig. 6 shows the result of the TTC estimation.

Figure 7. TTC estimation.

4.5. Time To Contact Method 2

The above first method to calculate TTC depends on the optical flow and FOE calculation, which can be rather noisy, depending on the actual environment the robot is in. As a result, the TTC estimate for different parts of the image is likely to have a high variance, making it difficult to use for the autonomous control of the robot. Alternatively, given the simple motion of the robot, i.e., it alternates between traveling in line segments and pivoting toward a new heading direction, we could use a simpler method for calculating TTC when the robot is moving forward. The TTC calculation is not performed when the robot pivots in place. Next, we describe the formulation of this method. First, given a point $[X, Y, Z]^T$ expressed in the camera frame (z-axis pointing forward) and its velocity $[U, V, W]^T$, its projection on the image plane is then $[\frac{fX}{Z}, \frac{fY}{Z}]$. Let the projection's velocity on the image plane be $[u, v]^T$, and we can derive the expression of u and v as follows:

$$\begin{aligned} u = \dot{x} &= f\frac{UZ - XW}{Z^2} \\ &= \frac{fU}{Z} - \frac{W}{Z} f\frac{X}{Z} \\ &= \frac{fU}{Z} + Cx \end{aligned} \tag{5}$$

where $C = -\frac{W}{Z}$ and is the essentially the inverse of TTC. In the case of negative W, as the robot approaches the object in the front, C would be positive and so is TTC, which is of the desired sign. Similarly, we have:

$$v = \frac{fV}{Z} + Cy \tag{6}$$

In the case of the robot moving forward, the velocity U and V are zero, and thus the above two expressions can be further simplified. Next, plug in u and v into the optical flow constraint equation (1), we have:

$$CxE_x + CyE_y + E_t = 0 \tag{7}$$

Since the above should hold for every pixel in the image, we could find C such that it minimizes the following:

$$\sum_{(x,y)} \left(C(xE_x + yE_y) + E_t\right)^2 \tag{8}$$

The C that achieves the minimum is $-\sum_{(x,y)} E_t(xE_x + yE_y) / \sum_{(x,y)} (xE_x + yE_y)^2$, and taking the inverse would produce the TTC estimate from the whole image. This pipeline can be implemented efficiently using vectorization, but it requires the camera calibration matrix to transform the pixel coordinates back to the image plane coordinate. We have thus carried out the camera calibration procedure, and the camera calibration matrix for our camera is the following:

$$\begin{bmatrix} 664.68 & 0. & 413.48 \\ 0. & 664.87 & 322.61 \\ 0, & 0. & 1. \end{bmatrix}$$

Thus to compute $[x, y]^T$ as needed in equation 8, one needs to use the homogeneous coordinate of each pixel and pre-multiply it with the inverse of the camera calibration matrix.

4.6. Depth Estimation

There are three methods to calculate depth information, known as Structure from Motion(SfM), Visual SLAM and using Optical Flow. In our project, we used the simplest method using Optical Flow. We have derived TTC from previous calculations, and then the next step is to obtain the translation vector of camera V_t. It should be noted that using images obtained by a single camera, the system can only determine the translational vector of the camera up to a scale:

$$D_r = D_i * e \tag{9}$$

where D_r is distance in the real world, D_i is distance shown in the images, and e is the scale factor between the real world and images. As a result, the final result we can get is a scaled diagram while we cannot obtain the depth information and absolute velocity in the real world. With

the computed TTC, we can obtain the depth image using the following equation:

$$X = V * T \tag{10}$$

where X is the scaled depth, V is the camera's scaled translational unit vector, and T is the TTC computed for each optical flow vector. Because we will show a scaled depth image, the translational unit vector of the camera movement does not influence the scaled depth image. To calculate the camera's translational unit vector, we could use matching points from subsequent images to calculate the essential matrix E. We can derive the translational vector of the camera by decomposing the essential matrix into the rotation part and translation part. The depth information is then the product of the translation vector with TTC, shown in the following equation. Then we can use the scaled depth image to show the depth information:

$$p = x_max - x/x_max - x_min * 255 \tag{11}$$

where p is the pixel value in the depth image. Then we can get the depth image, where the feature points of close objects will be bright while the feature points of objects further away will be dark.

Figure 8. Depth image. The white and gray dots correspond to the depth value calculated at these feature points.

4.7. Depth Estimation method 2

The first method is constrained by the input data, optical flow and TTC. Because we choose detecting corners of objects in the images, the depth images we derived from method 1 cannot include most features of objects. To detect more features and demonstrate their depth information, we decide to implement the second method SfM.The first goal of the method 2 is to reconstruct 3D scene. The steps are similar with method 1. We also need to match as much points as possible and calculate the Essential matrix. To obtain more matching points, we increase the edge threshold of sift algorithm. Eliminating edge responses thearom is from [2], we compute the principle of curvature by Hessian matrix H:

$$\left[D_xx, D_xyD_xy, D_yy\right] \tag{12}$$

Then we can compute the determinant of H:

$$Det(H) = D_xx * D_yy - (D_xy)^2 = ab \tag{13}$$

where a is the largest magnitude and b is the smaller one. Let $a = rb$

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{14}$$

The item $\frac{(r+1)^2}{r}$ is the minimum. Then we need let other points surpass the threshold r to eliminate edge errors. However, we need to input some points with low contrast in edge response, due to required large amount of matching points to reconstruct 3D scene.

Then we calculate projection and essential matrix after matching points. We can obtain 3D points by using projection matrix to triangulate 2D points. To check reprojection errors, the bundle Adjustment is necessary. Reproject 3D points to 2D points, we can check the errors between expected position. Then we can derive depth image using the scale method.

Figure 9. Depth image with method 2. The white and gray dots correspond to the depth value calculated at these feature points.

4.8. Balanced optical flow control

When an obstacle appears in one hemifield of the robot, it will create an imbalance in the optical flow magnitude across the left and right hemifield, as objects closer to the camera create a larger optical flow, assuming all objects are stationary while the robot moves. Therefore, one can create a control strategy that tries to balance the optical flow magnitude across the left and right hemifield. First, the optical flow index I is defined as follows:

$$I = \frac{\sum_{(i,j)\in \text{left}} m_{ij} - \sum_{(i,j)\in \text{right}} m_{ij}}{\sum_{(i,j)\in \text{left}} m_{ij} + \sum_{(i,j)\in \text{right}} m_{ij}} \quad (15)$$

where m_{ij} denotes the optical flow magnitude at pixel (i, j). The optical flow index I takes values between -1 and 1, given non-zero optical flow across the field of view. A simple control strategy can thus be defined below:

$$u_l + u_r = c \quad (16)$$

$$u_l - u_r = K_p \times I \quad (17)$$

where u_l and u_r are the control input for the left and right motors, K_p is a positive coefficient and c is a chosen positive constant. This control strategy will increase the left motor speed when the left hemifield has a larger optical flow magnitude, corresponding to the potential presence of an obstacle, and thus pivot the robot towards the right side and away from the obstacle. The opposite happens when the right hemifield has a larger optical flow magnitude. Therefore, this strategy will allow the robot to avoid obstacles residing in either hemifield.

5. Results

5.1. Testing the performance of different optical flow algorithm

To use optical flow information for efficient control of our robot, we first tested several implementations of different optical flow methods to see which one might offer the best result. The considerations include the speed with which the algorithms can be run on the onboard computer, i.e., Raspberry Pi 3B, and the suitability for the balanced optical flow strategy.

First, optical flow algorithms can be characterized by whether they are "sparse" or "dense". Sparse optical flow algorithms extract optical flow only for certain feature points while dense optical flow algorithms extract optical flow for every pixel in the image. As the balanced optical flow control algorithm depends on the optical flow in the left and right hemifield, a dense algorithm is preferable since it provides optical flow information beyond certain points. Nevertheless, the dense algorithms might be computationally more expensive than the sparse algorithms, and thus how fast the dense and sparse algorithms can be run on Raspberry Pi needs to be determined empirically. For the sparse method, we chose the Lucas-Kanade (LK) method, using our own implementation and the OpenCV built-in implementation. For the dense method, we chose the dense LK method and the Grunnar-Fauneback (GF) method (both are OpenCV built-in implementations). We first compared the time needed to process each frame (Table. 1), and sparse

Table 1. Execution time per frame for different optical flow algorithm

	LK (custom)	LK (OpenCV)	dense LK	Fauneback
Time (sec)	0.57±0.18	0.29±0.061	0.47±0.043	0.40±0.031

LK method with OpenCV implementation has the least execution time. The execution time for the two dense methods was comparable. Our own LK method implementation tends to run less efficiently and thus is excluded from further testing and implementation on the hardware. Next, we compared qualitatively the output from the remaining 3 algorithms by applying them to a video recorded when the robot follows a fixed rectangular trajectory (Figure 10). These methods offer quite qualitatively different results. The sparse LK method, as is based on feature points, heavily relies on distinct features in the images. As such, the floor optical flow is largely absent. On the contrary, the dense LK and GF can more reliably extract optical flow related to the floor. Comparing the dense LK method with the GF method, GF seemed to be better in extracting optical flow further in the distance, while the former mostly extracts optical flow closer to the robot. Therefore, for the purpose of obstacle avoidance, the GF method is the most suitable, as it offers the ability to detect optical flow further down the robot's path and the computation time is less than the dense LK method (Table. 1). The video demonstration of the 3 algorithms applied to the recorded image sequence can be found with the following hyperlinks (sparse LK, dense LK, Fauneback).

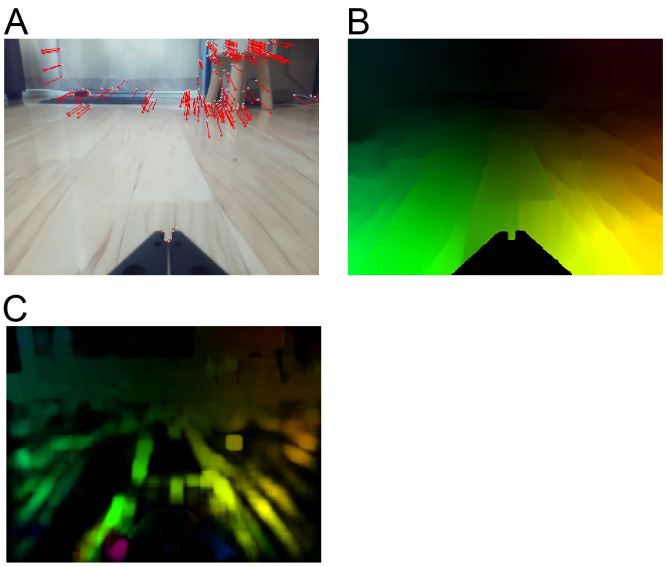


Figure 10. Example optical flow output of 3 different algorithms. (A) Sparse Lucas-Kanade method. (B) Dense Lucas-Kanade method. (C) Grunnar Fauneback method.

5.2. Balanced optical flow control

To further test the feasibility of the balanced optical flow control, we calculated the optical flow magnitude in the left and right hemifield using a video sequence recorded driving the robot in a straight line while a known obstacle is present in the left hemifield (Figure 11). With this experiment, we confirmed that the obstacle in one hemifield will cause the optical flow in the corresponding hemifield to increase, and thus the control strategy is valid.

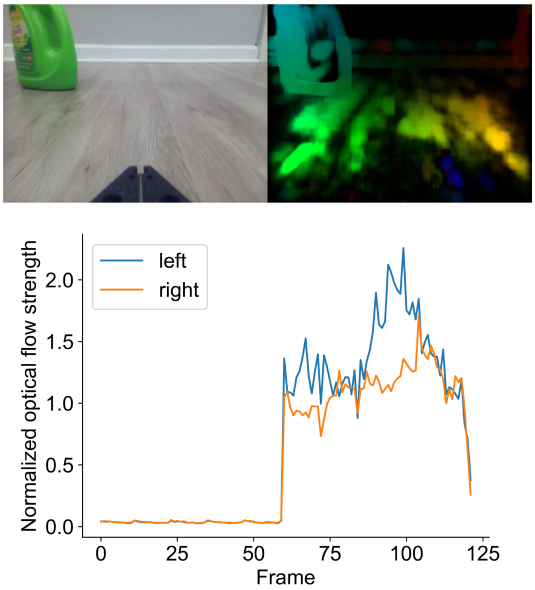


Figure 11. Gunnar-Farneback method applied to one footage where the robot is approaching an obstacle (green detergent bottle).Top: an example frame of the actual video footage and the GF method output. Bottom: left and right hemifield optical flow magnitude as a function of frames. The robot started moving around frame 60. The left hemifield had a larger optical flow magnitude.

Next, we applied the control strategy outlined in section 4.8, and we show here that our robot can pivot away from obstacles residing on either the left or the right side of the robot. Two video demonstrations can be found with the following hyperlinks (obstacle avoidance video 1, obstacle avoidance video 2). Together, we show and confirm that the balanced optical flow control strategy is valid and applicable to obstacle avoidance. However, this strategy would fail if the obstacle is directly on the forward path of the robot, where it will create a roughly balanced optical flow magnitude across both the left and right hemifield. In this case, we relied on the Time to Contact (TTC) to make additional control decisions for the robot.

5.3. Performance of TTC

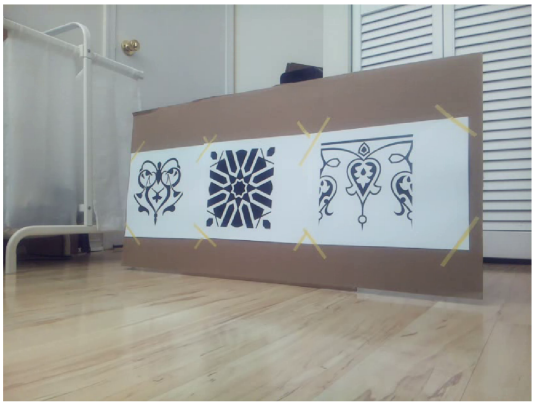
In section 4.4, we demonstrated the first method to calculate TTC by using the distance of feature points from the FOE and the translational velocity. A video demonstration can be found with the following hyperlink (TTC video). As the video shows, feature points that are not extracted from the obstacles could lead to abnormal TTC. Furthermore, the noise in the FOE calculation that happened frequently also causes the TTC to biased values.

5.4. Augmented control strategy using TTC

For the actual robot's control, TTC is calculated using the second method outlined in section 4.5 as it can be more efficiently implemented and less computationally complex (no feature points need to be extracted). In addition, as the second method does not rely on the FOE calculation, which is often very noisy, it does not suffer from additional complexity introduced by inaccurate FOE.

In the case of approaching obstacles creating roughly equal optical flows across the left and right hemifield, e.g., walls, a decision to first turn left by a random amount is made if the calculated TTC drops below some preset threshold for the current frame. If not, then the default balanced optical flow control is used instead. The tuning by the robot will steer itself away from obstacles such as walls or directly in front of its path. To test the feasibility of using the current TTC calculation for the above-mentioned strategy, we performed some experiments where the robot approached a cardboard wall while a video was recorded (TTC testing video). We then calculated TTC offline and we observed that the TTC value indeed decreased as the robot approached the cardboard wall (Figure 12), and in this example, a threshold of 20 would be suitable to make the robot pivot and turn to a new heading direction. These results suggest that our augmented strategy would be feasible.

We thus combined the TTC control with the balanced optical flow control, and we carried out experiments in an arena enclosed by cardboard walls lined with printed patterns (Figure 13). The robot was capable of roaming within the arena while avoiding crashing into walls and the obstacles placed within the arena. The full videos of the two demonstrations can be found using the following hyperlinks (arena run video 1, arena run video 2). During these demonstrations, we also recorded the distance to the wall using the onboard ultrasonic sensor, which could provide some validation of our TTC calculation (Figure 14). Our result shows that the TTC value was significantly and positively correlated with the distance measure (p-value<0.001). Therefore, although the TTC calculation can be noisy, it did covary with the actual distance to the obstacles, thus TTC is a valid approach.



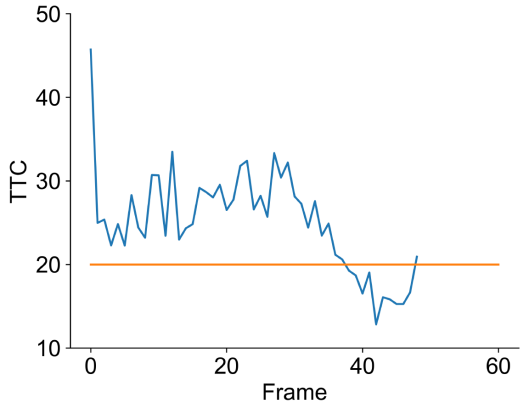


Figure 12. TTC value as the robot approached a cardboard wall. Top: the camera image while the robot drove toward the cardboard wall. Bottom: TTC calculated as a function of the frame. Its value decreased as the robot approached the wall. Frame 0 is the frame in the video the robot started moving forward. The horizontal orange line represents a potential threshold of 20, where the robot would pivot away from the way.

Figure 13. Two demonstrations where the combined TTC and balanced optical flow control were used while the robot wandered in the arena. Top: a single obstacle was placed in the arena. Bottom: two obstacles were placed in the arena.

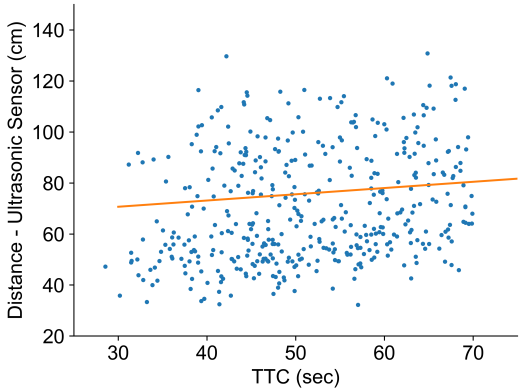


Figure 14. TTC calculation during the arena demonstrations is plotted against the simultaneous distance measures collected using the onboard ultrasonic sensor. TTC calculation is positively correlated with the distance measure within the range shown, i.e., Distance $= 0.24 \times$ TTC $+ 63.3$, p-value<0.001

6. Conclusion

In this project, we implemented an optical flow based algorithm for obstacle avoidance on a mobile robot and we show that the robot can avoid obstacles, relying on the single camera equipped. In the process, we tested different optical flow algorithms and implemented calculations such as Focus of Expansion, Time-to-Contact and depth map. Overall, we show that this rudimentary control strategy based solely on the optical flow is quite effective, but it can benefit from improved robustness of the calculation such as TTC.

7. Future Work

We could extend this project first by finding more robust algorithms for estimating TTC. Our first method for calculating TTC seemed to have a high variance, while the second method can also produce noisy results, depending on the particular environment. Therefore, a more robust method is called for as the TTC constitutes the very first decision the robot makes in the entire autonomous decision-making process. Second, our current methods assume a stationary environment, and thus implementing a control strategy suitable for a dynamic environment with moving obstacles can be a natural next step.

8. Acknowledgement

For this project, Yi-Chung Chen is responsible for implementing the LK algorithm, TTC calculation method 2, and making the pipeline flowchart. Vamshi Kalavagunta is responsible for implementing the FOE calculation. Weili Su is responsible for implementing the TTC calculation method 1. Zidong Zhao is responsible for implementing the depth map calculation. Ji Liu is responsible for implementing the TTC calculation method 2, implementing the control

algorithms on the robot, and testing and taking videos with the robot.

For the report, Yi-Chung Chen wrote section 4.2 and 4. Vamshi Kalavagunta wrote the abstract and the section , 1, 2, 3, and 4.3. Weili Su wrote section 4.4 and 5.3. Zidong Zhao wrote 4.6,4.7. Ji Liu wrote section 4.1, 4.5, 4.8, 5.1, 5.2, 5.4, 6, and 7. For figures and tables, Yi-Chung Chen made Figure 1, 3, and 4. Vamshi Kalavagunta made Figure 5, 6. Weili Su made Figure 7. Zidong Zhao made Figure 8 and 9. Ji Liu made Table 1, Figure 10, 11, 12,13, and 14.

References

[1] Souhila K., Karim A., Optical Flow Based Robot Obstacle Avoidance. International Journal of Advanced Robotic Systems 2007; 4(1): 13-16.

[2] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.