ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

OPTIKAI ÁRAMLÁSRA ALAPOZOTT AKADÁLYKERÜLÉS INTEGRÁLÁSA TÉRINFORMATIKAI NAVIGÁCIÓVAL

Szerző(k): **Nás Hunor István**
mérnökinformatikus BSc. szak, III. évf.
Nás Levente Gellért
mérnökinformatikus BSc. szak, I. évf.

Konzulens(ek): **Balázsné Dr. Kail Eszter**
adjunktus

Budapest, 2022.

Contents

1 Abstract

Everything in the technology field is advancing towards automation; whether that is full or semi automation. Obstacle avoidance falls under these automation efforts. However, the sensors used to achieve reliable results are very costly. This research aims to develop and test a reliable optical flow based autonomous obstacle avoidance algorithm. Developing a refined and optimized optical flow algorithm would enable further obstacle avoidance research without the limitation of cost. Furthermore, we intend to integrate the obstacle avoidance algorithm into a geo-spatial navigation algorithm to create a closed loop rover-like vehicle capable of land surveying, collecting vital sensor data on large monotone landscapes, monitoring gas pipeline and electrical grid stations, and surveying other tedious and repetitive land masses autonomously. We will test the accuracy and reliability of the obstacle avoidance by feeding collected test frames into the algorithm and analyzing the output results. To test more realistic situations, a rover equipped with a raspberry pi, electric brushless motor, two electric servos, camera, and a proximity sensor will be built to test the algorithm dynamically. The geo-location navigation algorithm will be tested separate from the obstacle avoidance algorithm, utilizing the rover build, by following a predetermined path of coordinates. The path following accuracy will be analyzed for reliability. Throughout the paper, we will unpack the ideas of geo-spatial navigation and autonomous obstacle avoidance in the context of optical flow.

2 Introduction

Automation has been the most important technological step in any domain of service. Factories quickly adopted automation to enhance their output without compromising the quality and cost of their product. Today's technology has made it possible to integrate autonomous decision making into any platform interacting with humans, even high risk machines like automobiles. Obstacle avoidance plays a huge role in creating a reliable autonomously driven vehicle. Expensive sensors such as lidar, radar, ultrasonic sensors and cameras are required for such tasks to reduced the possibility of error and maximize safety. Most applications require a hybrid approach to effectively perform at a consistent safety standard. Since the end product is a rover-like surveyor, it would be optimal to leave out such complex hardware without losing accuracy and reliability. By making the two-frame motion estimation algorithm [6] run efficiently on a micro-controller there is an opportunity to normalize autonomous navigation on smaller scale vehicles like rovers. Adding a sufficient navigational system alongside the autonomous obstacle avoidance system would yield a very useful, cost effective and readily available structure. To make this happen, a reliable model capable of weighing the navigational data according to the obstacle avoidance data is required. This project aims to create such system and provide a proof of concept of the technology.

2.1 Autonomous Vehicle Technology

Autonomous vehicle technology and concepts have been in development ever since computers were small and strong enough to allow for a realistic solution. The idea of autonomous driving is quite simple: assign vehicular control and decision making to a computer. Though some individual components of driving have been automated for quite some time (adaptive cruise control, lane assist, engine control, etc.), full autonomy is the desired technology. The steering, acceleration, braking and any manual on board system functions that are usually controlled by a human chauffeur would now be under the command of a computer. The complexity of automating a simple task such as driving is enormous. Road and weather conditions are extremely dynamic, objects in the vehicle's surroundings are consistently moving and each vehicle operator is independent from the other; pedestrians

and other drivers are unpredictable. To overcome such hardships sensors such as Light Detection and Ranging (lidar), Radio Detection and Ranging (radar), Sound Navigation and Ranging (sonar), infrared cameras, stereovision (depth perceptive vision) and other vision mimicking sensors (cameras) are used to collect data about a vehicle in space [24]. In short, the data collected by these sensors is then aggregated and used to calculate the next step the computer should take to safely advance the vehicle along its route. Autonomous vehicles have the potential to make driving safer and more efficient.

2.2 Geo-spatial Navigation

Similar to how sensors help a computer decide its surroundings and position in space, there are sensors capable of communicating with the Global Navigation Satellite System (GNSS) to determine the precise geographical location of an object. There are four main GNSS constellations available: Global Positioning System (GPS) developed by the United States of America, Global Orbiting Navigation Satellite System (GLONASS) developed by Russia, Galileo developed by the European Union and BeiDou developed by China. There are other regional constellations too, but for the sake of simplicity, they will not be mentioned. These systems consist of satellites orbiting the Earth, maintaining a network of navigational data communication. There are very expensive GPS sensors with extremely high accuracy and cheaper sensors with less precise accuracy [24]. Without the proper navigational data, an autonomous vehicle is useless. Geo-spatial navigation is an important staple to the suite of sensors that must be on-board a vehicle.

2.3 Obstacle Avoidance

Vehicle obstacle avoidance is the process of avoiding objects that present themselves in the path of a vehicle as it moves forward. Furthermore, autonomous obstacle avoidance allows for a computer to make the proper adjustments to fulfill the obstacle avoidance criteria described above.

2.3.1 Brute Force Obstacle Avoidance Method

Of course it is quite simple to physically detect an obstacle, but it is not so trivial to figure out the direction to apply the steering adjustment to avoid the obstacle. Assuming there is a sensor that detects obstacles, the brute force obstacle avoidance method would be to steer right or left when an obstacle is sensed, move left or right for a couple seconds, reevaluate the obstacle situation and adjust the steering angle based on updated obstacle information. This method may work, however, it is inaccurate and not reliable because the effectiveness of the algorithm depends solely on the sensors field of view.

2.3.2 3D Dynamic Obstacle Avoidance Method

A more dynamic algorithm would detect the obstacle, evaluate the environment and determine the most optimal direction and angle to steer in. This is hard to determine with a single sensor. If the criteria would be to use a single sensor, lidar and radar sensors would be the realistic options. In short lidar and radar sensors emit laser light and sound waves, respectively, and measure the amount of time it takes to receive the returning light rays or sound waves. This time measurement can then be used to determine the position of objects around a sensors environment, resulting in a point cloud map. These point cloud maps can then be used to determine the presence of an obstacle along with the direction of steer and the steering angle itself. Though these sensors are extremely reliable and effective, they are complex. Complexity makes the sensors hard to use, hard to process output data

and expensive. Alongside laser sensors and sound sensors, there are stereoscopy capable cameras that match two camera streams to create a 3D output and time-of-flight (TOF) cameras providing 3D information with a real-time frame rate, but these sensors are also very expensive and hard to handle [12].

2.3.3 Alternative 2D Dynamic Obstacle Avoidance Methods

All aforementioned methods try to utilize three dimensional information to avoid obstacles. To mitigate the financial obstacle in such an area, two dimensional camera data can also be used to detect the presence of objects and sensor surroundings. The effectiveness of such methods is reduced, but the vast availability of the sensors makes up for the lack of effectiveness.

2.3.4 Block-Based Motion Estimation

There are three methods that seem viable. The first method is called block-based motion estimation (BBME) [12]. In BBME, a video image sequence is split up into individual frames, which is then divided into non-overlapping blocks of pixels. The motion vector of a block is then calculated from one frame to the other by minimizing the predefined searching criterion, commonly the Sum of Absolute Differences (SAD). The determined best matched blocks are then used to calculate the spatial difference. Please refer to the cited paper on the mathematical specifics and precise algorithmic steps.

2.3.5 Difference Map Method

The second viable method is a hybrid solution utilizing ultrasonic sensor and expectation maps [3]. The ultrasonic sensor is used to detect moving obstacles and the vision-based technique is used to detect stationary objects. To create the expectation maps, the robot must be manually driven through the terrain (in this case an office-like floor) with any obstacles cleared out of the way. Images taken of the cleared terrain are then rendered to create an expectation map. When the robot drives through the same terrain it records the images at a given location, applies an edge detection operator resulting in an edge map. It then constructs a difference map from the differences between the expectation and edge maps and then calculates the proper direction to proceed. The calculation is simplified here. Please refer to the cited paper for the precise explanation.

2.3.6 Optical Flow Method

The third method would be to calculate the optical flow of picture frames taken in succession and determine steering angle and presence of obstacles based on the optical flow of the image [2]. This research focuses on a rich algorithm that relies solely on a camera sensor, a Raspberry Pi camera to be precise and a simple proximity sensor. Please refer to section (3) for optical flow information. The proximity sensor will detect the presence of an obstacle, after which the latest captured frames will be processed by the optical flow algorithm. The optical flow method is only viable and tested with stationary objects due to the noise when moving objects are introduced. The resulting angle of steer will then be applied to the servos and the rover will move on. Optical flow is a popular method used for obstacle avoidance and has been applied to rover-like vehicles [2] and Unmanned Aerial Vehicles (UAV) [4]. Please refer to these citations for specifics.

3 Optical Flow

Optical flow is the comparison of the movement of pixels of two two-dimensional images taken in succession. Two pixel qualities are analyzed to determine object motion: brightness consistency and spatial smoothness [14]. Brightness consistency deals with pixel brightness from one frame to the next, while spatial smoothness deals with a pixel belonging to the same surface as the previous frame. There are two main methods to evaluate these images taken in succession: sparse optical flow and dense optical flow.

3.1 Sparse Optical Flow

Sparse Optical Flow algorithms track the movement of specific pixels. These specific pixels are determined using algorithms like the Harris corner detection algorithm [9] or the Shi-Tomasi corner detection algorithm [17]. Usually these specific pixels are concrete objects like cars on the road or people on the street. Sparse calculations are quick when compared to dense optical flow calculations. It is important to note that sparse optical flow algorithms only produce flow vectors for the points deemed important by a object detection algorithm.

3.1.1 Kanade-Lucas-Tomasi Method

The Kanade-Lucas-Tomasi (KLT) method is a sparse optical flow algorithm utilizing the Lucas-Kanade [13] (see section 3.2.1 for a more thorough explanation of the Lucas-Kanade Method) optical flow algorithm and the Shi-Tomasi corner detection algorithm [17]. It is important to note the core ideas behind this method, since it utilizes a dense optical flow algorithm alongside the principles of feature tracking over time [21]. Though this particular sparse optical flow algorithm relies on a dense optical flow algorithm, it is not a dense optical flow algorithm due to its specialized calculations.

3.2 Dense Optical Flow

Dense Optical Flow algorithms track the movement of all the pixels in two given frames resulting in a flow vector for each pixel. Of course it is assumed the transition from one frame to the next is ideal. There are many methods that fall under dense optical flow, which will be discussed very briefly. Please refer to the citation for a more extensive explanation. The method utilized in this project and explained in depth, is a relatively modern Gradient-Based Estimation deemed quickest and most precise [14].

3.2.1 Region-Based Matching

Region-Based Matching maximizes normalized cross-correlation of shifts in images or minimizes a distance measure using sum-of-squared difference method [1]. This method tends to be inaccurate due to noise.

3.2.2 Phase-Based Estimation

Phase-Based Estimation is used to estimate the offset of pixels between two images. The images relies on the frequency domain and rely heavily on Fourier Transforms and Gaussian filter smoothing [1].

3.2.3 Gradient-Based Estimation

When the first image is taken, every pixel in the captured image has an intensity I. The intensity is said to be a function of x, y, and t considering a certain amount of time has passed between the capture of the two images; that is,

$$I(x, y, t). \tag{1}$$

As the second image is taken, assuming the capturing device is moving, the same pixels that had an intensity can be observed in the second image with the same intensity. Of course it is assumed that the pixels did not experience noise and each pixel from the first image stayed in the frame of the second image. For any arbitrary pixels movement there is a change in x, y, and t.

$$I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2}$$

Since we are taking into consideration two images taken with minimal Δt, we can assume the movement of a given pixel from the first image to the second image to be very small. The $I(x, y, t)$ function can be expanded using Taylor Expansion, using the Basic Gradient-Based Estimation [7],

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + ..., \tag{3}$$

which by linearization of equation (3) yields

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0. \tag{4}$$

When both sides of the linearization is divided by Δt, equation (4) results in

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0. \tag{5}$$

Rearranging the equation yields

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v = -\frac{\partial I}{\partial t}. \tag{6}$$

Since u and v are the x and y components of the optical flow of $I(x, y, t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the change in the image at (x, y, t), we can write equation (6) as

$$I_x V_x + I_y V_y = -I_t. \tag{7}$$

Equation (7) has two unknowns, V_x and V_y, with only one equation. This is known as the aperture problem [25]. A separate method must be used to determine these values.

Lucas-Kanade Method

In the Lucas-Kanade method it is assumed that the flow of pixels is constant in an arbitrary neighborhood around a pixel of concern. The optical flow equations for all pixels is then estimated in the neighborhood using a type of Newton-Raphson iteration [13]. The Lucas-Kanade method assumes a minimal displacement of pixels from one frame to the next, thus assuming a small displacement

of the neighborhood of pixels as well. Equation (7) must hold for all points in the neighborhood centered at point p,

$$\begin{aligned} &I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ &I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ &I_x(q_3)V_x + I_y(q_3)V_y = -I_t(q_3) \\ &\vdots \\ &I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n), \end{aligned} \tag{8}$$

where $q_1, q_2, q_3, ..., q_n$ are the pixels inside the neighborhood. To simplify the form, we can express the system of equations in matrix form $Ax = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ I_x(q_3) & I_y(q_3) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \; x = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \; b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ -I_t(q_3) \\ \vdots \\ -I_t(q_n) \end{bmatrix}.$$

The system of equations has significantly more equations than unknown variables. It is now possible to obtain a compromised solution using the least squares principle. For more accurate results, the pixels in the neighborhood are weighted based on how far or close a given pixel is to the center pixel p.

Horn-Schunck Method

The Horn-Schunck method [10] assumes smoothness in the flow [14]. The goal is to minimize the global energy function

$$E = \iint \left[(I_x V_x + I_y V_y + I_t)^2 + \alpha^2 (||\nabla V_x||^2 + ||\nabla V_y||^2) \right] dx\, dy \tag{9}$$

as much as possible. This is done utilizing equation (7), a regularization constant α, which the larger it gets the smoother the flow, and solving the associated multi-dimensional Euler-Lagrange equations [10].

Gunnar Farnebäck Method

The Two-Frame Motion Estimation by Gunnar Farnebäck [6] utilizes polynomial expansion along with displacement field estimation from the acquired coefficients to predict motion of objects given two frames. Firstly a polynomial expansion is taken of a neighborhood of pixels. Assuming the polynomial undergoes an ideal translation, (that is any given pixel from the first frame moves a fixed amount of units in the x and y direction to the second frame) we consider the exact quadratic polynomial

$$f_1(x, y) = p_1 x^2 + 2r_1 xy + q_1 y^2 + s_1 x + t_1 y + u_1. \tag{10}$$

This quadratic polynomial can be expressed in terms of matrix $A_1 = \begin{bmatrix} p_1 & r_1 \\ r_1 & q_1 \end{bmatrix}$, vector $b_1 = \begin{bmatrix} s_1 \\ t_1 \end{bmatrix}$, vector $x = \begin{bmatrix} x \\ y \end{bmatrix}$ and scalar $c_1 = u_1$ as

$$f_1(x) = x^T A_1 x + b_1^T x + c_1. \tag{11}$$

Using linear algebraic methods, we can represent $f_1(x) = x^T A_1 x + b_1^T x + c_1$ as

$$f_1(x) = A_1 x^2 + b_1 x + c_1. \tag{12}$$

Translating the original signal, by $d = \begin{bmatrix} u \\ v \end{bmatrix}$ and evaluating it in the form of equation (12) results in

$$\begin{aligned} f_2(x) = f_1(x-d) &= A_1(x-d)^2 + b_1(x-d) + c_1 \\ &= A_1 x^2 - 2A_1 dx + A_1 d^2 + b_1 x - b_1 d + c_1. \end{aligned} \tag{13}$$

Mapping the resultant quadratic back to form (11) yields

$$\begin{aligned} f_2(x) &= x^T A_1 x + (b_1 - 2A_1 d)^T x + d^T A_1 d - b_1^T d + c_1 \\ &= x^T A_2 x + b_2^T d + c_2 \end{aligned} \tag{14}$$

where

$$\begin{aligned} A_2 &= A_1 \\ b_2 &= b_1 - 2A_1 d \\ c_2 &= d^T A_1 d - b_1^T d + c_1. \end{aligned} \tag{15}$$

Assuming A_1 is non-singular we can solve for the value of translation d,

$$\begin{aligned} 2A_1 d &= b_1 - b_2 \\ d &= \frac{A_1^{-1}(b_1 - b_2)}{2}. \end{aligned} \tag{16}$$

Assuming the resultant signal is a polynomial and a global translation relating the two signals is unrealistic, however, if errors are minimized then equation (16) and its relation can be used for real functions. Taking a polynomial expansion on first image of two gives us expansion coefficients $A_1(x), b_1(x)$, and $c_1(x)$ and expansion coefficients $A_2(x), b_2(x)$, and $c_2(x)$ for the succeeding image. The global polynomial in equation (11) should be replaced by the above mentioned local polynomials. In an ideal situation this should mean $A_1(x) = A_2(x)$ but ignoring assumptions

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{17}$$

is the reasonable approximation. Equation

$$\Delta b(x) = \frac{1}{2}(b_1(x) - b_2(x)) \tag{18}$$

is introduced to obtain the primary constraint

$$A(x)d(x) = \Delta b(x), \tag{19}$$

where the global displacement d from equation (14) is replaced by a spatially varying displacement field $d(x)$. The precision of the algorithm is further improved but irrelevant for basic understanding of this method.

4 Optical Flow Method implemented

The method used in our project was proposed by Gunnar Farnebäck. OpenCV has implemented and optimized this algorithm; it is realistic to calculate optical flow of incoming real time frames. This method is also the most precise and quickest method when compared to the Lucas-Kanade and Horn-Schunck methods [14].

5 Hardware Used to Build Rover

The RC car chassis used is an HPI Savage 4.6 chassis [20]. The micro-computer running the main calculations is a Raspberry Pi 4b with 2GB of RAM [8], along with a 5MP Rsapberry Pi camera. The servos and batteries are HPI proprietary hardware that came with the RC car. The motor is a 3800KV Surpass Hobby 1:12 Brushless Sensorless Motor [5]. The board used for GNSS information and calculation is a Sony Spresense IoT (Internet of Things) board [18]. A 5-40V input 5V ouput DC to DC converter was also used to power the closed loop system [19].

5.1 Raspberry Pi

The Raspberry Pi board is the hub of all the electronics and the central power source. This is where all the software, calculations and drive scripts are executed. Once the Pi calculates the angle of steer, it sends a PWM (Pulse Width Modulation) signal to the servo to change the position. The motor speed is also controlled by a PWM signal that the Pi provides. The Spresense board is connected to the Pi through a serial communication interface and information generated by the Spresense board is received by a Python script on the Pi. The camera connects directly to the Pis camera interface feeding the camera data directly to the Pi. The power supply of the Pi comes from a 7.2V (Volt) 4700MaH (Milliamp Hour) rechargeable Nimh (Nickel-Metal Hydride) battery connected to a step down DC (Direct Current) to DC converter. The Pi is running a Raspbian operating system to avoid compatibility and resource management problems. To avoid overheating the Pi, a larger cooling fan is used to dissipate the heat.

Pulse Width Modulation Pulse Width Modulation (PWM) is the process of sending a high or low pulse signal with a given width at a high rate [16]. A servo or motor Electronic Speed Controller (ESC) has its power and ground pins along with a control signal pin. When this control pin receives a high pulse with a given width, it commands the hardware to rotate or spin at a given angle or speed. PWM makes it very simple to turn a servo a certain angle and to run an electric motor at a certain constant speed.

Pigpio Library The pigpio Rapsberry library was used to control the servos and the motor [15]. This library allows control of the general purpose input output (GPIO) pins through Python code. Any GPIO pin can be configured to listen or send PWM signals, allowing for the ease of communication between the Pi and the respective hardware.

5.2 ICE Tower Fan for Pi

To maximized the operation speed of the Pi, it was essential to maximize the boards cooling capability. The ICE tower fan is a active air cooling fan designed to keep the Raspberry Pi board electronics cool [11]. Heat sinks didn't cool effectively enough due to their passive nature, this is why this cooling method was chosen.

5.3 Sony Spresense Board

The Sony Spresense IoT makes it possible to collect and process GNSS data [18]. Initially the Spresense board was going to run the main software. However, the on-board processor specifications limited the boards ability. Running the software was pivoted to the Pi and the Spresense function strictly became GNSS focused.

5.4 Components

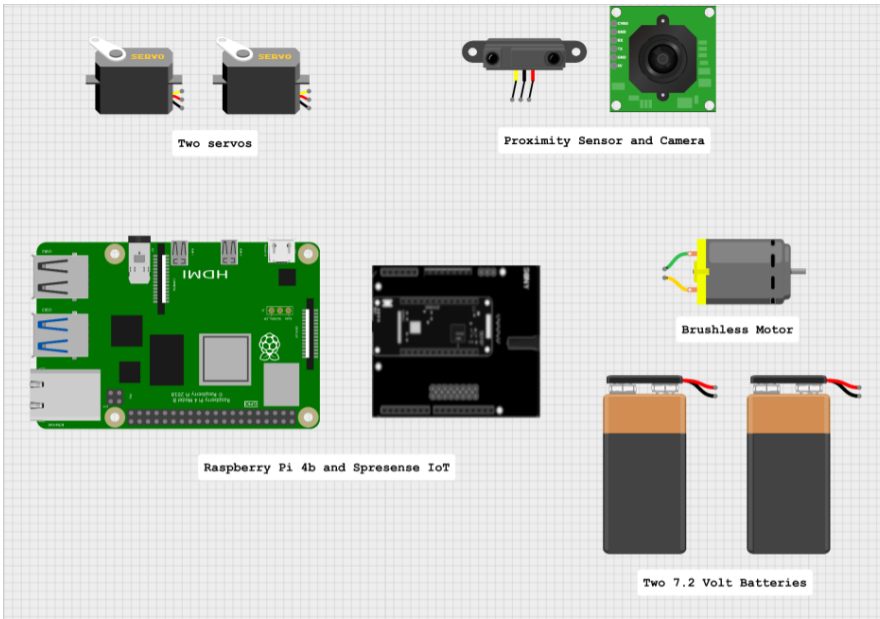


Figure 1: Components Diagram

6 Code Snippets

6.1 Implementation With Code

Two important entities must work together for the correct functioning of the rover: the hardware and the image processing. The camera, proximity sensor, motor and servos are all easily controlled by simple Python code. Because of this, a main Python script is used to connect the hardware and the C++ software. The Python script collects the frames, runs the C++ code through the generated executable file after building and applies suggested changes to the rover's servos real-time.

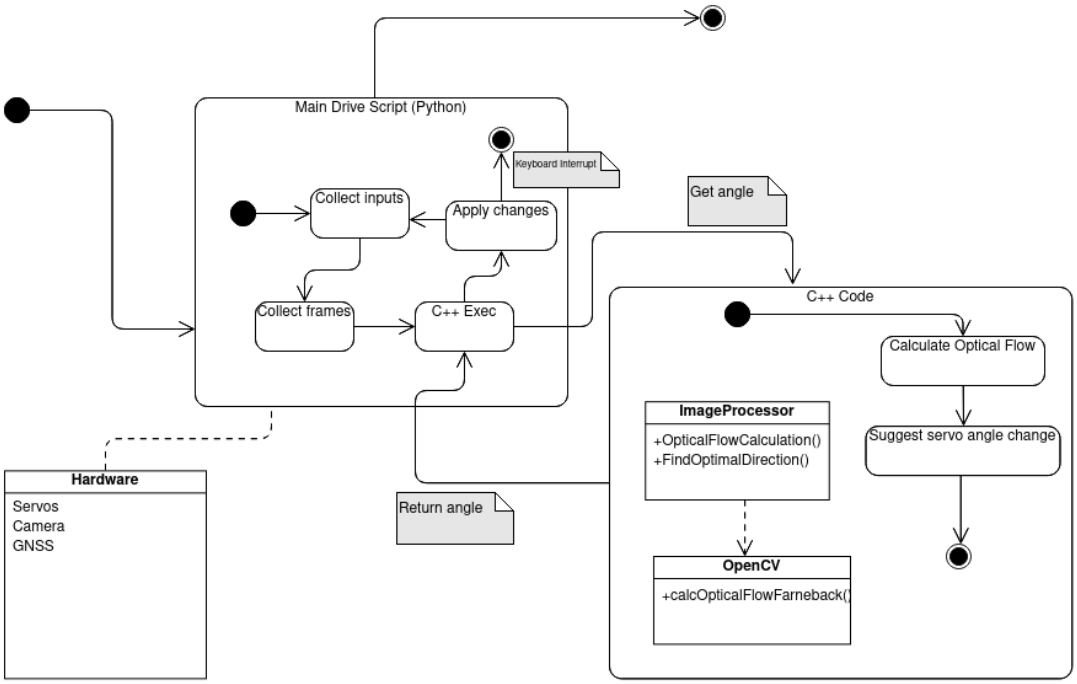


Figure 2: Code Base Diagram

The C++ code receives the input frames the script collected and executes the necessary image processing functions. A separate image processor class utilizes OpenCV's various methods to accomplish these goals. By calculating the optical flow between the two input images using the Gunnar Farnebäck optical flow algorithm and then analyzing the resulting optical flow, the software can accurately suggest a change in trajectory. The algorithm which calculates the change in angle the rover must make in order to avoid an obstacle partitions the optical flow matrix into sub-matrices. The size of the sub-matrices can be easily configured by the user. Once sub-matrices are created, the sum of the magnitude of optical flow in each sub-matrix is calculated, and the window or sub-matrix with lowest sum is the direction the rover should go in. The lower the optical flow sum, the less movement in that sub-matrix, and hence, the rover should follow that direction. The partitioning algorithm was designed using functional programming elements in C++. To improve the speed at which these calculations are done, these optical flow and partitioning calculations are only run if the proximity sensor detects obstacles that may need to be avoided.

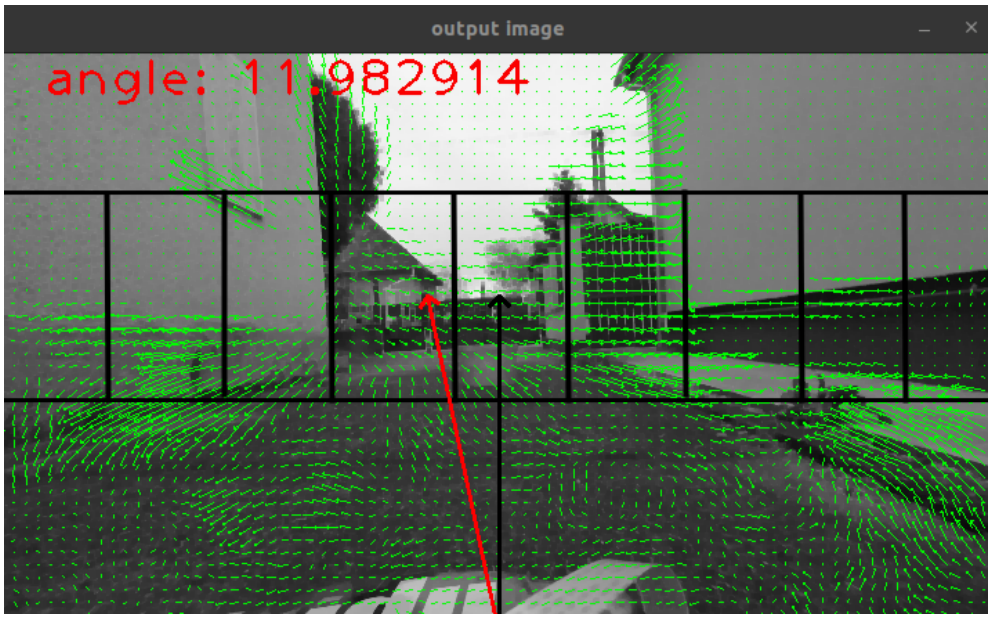


Figure 3: Example of partitioning

Algorithm 1 Partitioning algorithm

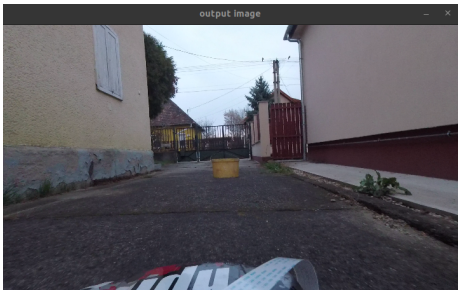
```
Matrix opticalFlow
int minSum
int direction
int size
fmap rowCrop
fmap transpose
for i = step, 2 * step,... rows(opticalFlow) do
    float magnitude ← GetSum(rowCrop, transpose, i, opticalFlow)
    if magnitude < minSum then
        direction ← i
    end if
end for
```

6.2 GNSS Spresense Output

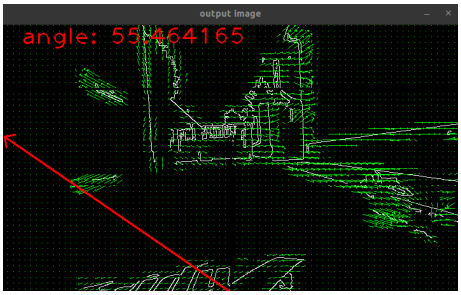
The GNSS-based navigational algorithm was not developed fully. The Sony SPresense Board provided the navigational data to make such a feature possible. A python script running on the Raspberry Pi was capable of getting the longitude, latitude, heading and velocity provided by the SPresense board hardware, however, the accuracy of the information was poor due to the sensor limitations. The positioning information had an error of ± 2-5 meters. The idea was to calculate to bearing using the current position of the rover and the predetermined path of coordinates [23]. After the bearing was accurately calculated, the next step would have been to subtract the current heading of the rover from the calculated bearing to determine the direction the rover must steer to get to the desired position [22]. Drift in this case was not a factor since the rover was rolling on

hard terrain. If the difference between these two angles results in a negative value, the rover should steer left, and if the difference is positive the rover should steer right. If the difference is close to zero then the rover has reached the first desired coordinate on the predefined path and can move toward the next defined coordinate. Due to lack of time and sensor inaccuracy, this algorithm could not be tested and refined.

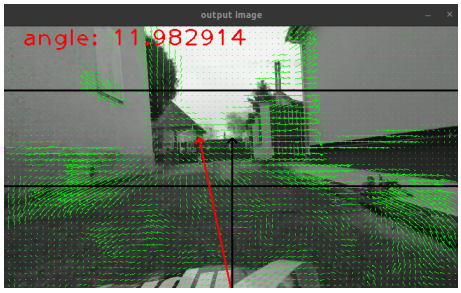
7 Testing



(a) Bucket obstacle



(b) Filtering with edge detection



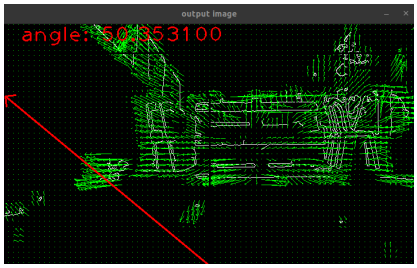
(c) Filtering with erode and dilate

Figure 4: Bucket Images

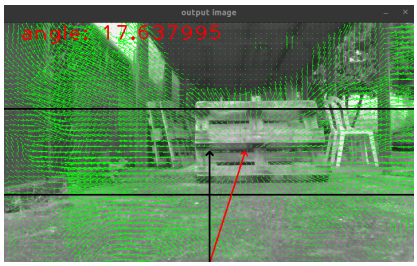
The first testing environment was an obstacle (bucket) in a clear corridor (Figure 4a). It is important to note the rovers velocity was approximately 1 kph (approximately .2 m/s). There was no exact measurement of the velocity of the rover with a precise tool, only the SPresense boards velocity measurement ability. The tests were run with two different filtering techniques: dilation coupled with eroding (Figure 4c) and blur coupled with edge detection (Figure 4b). Image 4b represents the optical flow output using edge detection. A problem with this technique immediately emerges: solid colored walls are not recognized. The point of edge detection algorithms is to identify drastic changes in the image gradients. Those drastic changes represent edges. Any solid colored wall or surface will not be accounted for when the rover calculates a new trajectory. In reality, the rover ran into this exact problem with this testing environment. With solid colored walls on either side, when a new path was to be calculated, the rover did not account for the wall and suggested a direct collision with it at 55.46 degrees. The edge detection software can be utilized with different parameters to achieve more detailed frames, but this also sheds light on the unnecessary features in the frame that were to be filtered out initially.

Similar to edge detection, dilation and erosion work to remove noise and unnecessary details our optical flow algorithm does not want to track. Contrary to edge detection, however, is the amount of detail still left in the image. Image c represents the test run of the same environment with dilation and erosion instead of edge detection. The optical flow algorithm was able to track the wall with a little more precision, giving it just enough optical flow for the rover to know that it is an obstacle/wall. With this in mind, the on-board computer was able to calculate a new path that directly avoided the bucket. The solid colored walls were still not ideally detected, but it was a significant improvement from just edge detection.

(a) Pallet Obstacle



(b) Filtering with edge detection



(c) Filtering with erode and dilate

Figure 5: Pallet Images

The second testing environment was a lot more chaotic. There was a direct obstacle (pallet) in front of the rover, but many side objects were also around it (Figure 5a). Figure 5b represents the test run with edge detection as filtering. Since the environment is so chaotic, the edge detection was very successful in isolating the objects that could cause a collision. However, similar to the first testing environment, the edge detection failed to identify the wall and fencing to the left of the rover. As a result, when calculating a new path, the on-board computer suggested this direction. The same issue prevailed.

Contrary to the first testing environment, the dilation and erosion filtering was no improvement. The chaotic environment lead to a very complicated and chaotic optical flow map (Figure 5c). With the abundance of optical flow vectors, the algorithm was not able to make any intelligent decisions with the new path. In complicated environments, neither filtering method proved to be efficient.

Although both filtering techniques had some sort of flaw, the two seem to compliment each other when used in unison. The dilation and erosion "sees" the solid colored details of a wall, while the edge detection can isolate obstacles in chaotic environments. A generalized filtering algorithm could be designed with a combination of filtering techniques, reeking the benefits of each technique.

8 Future Works

8.1 Improvements

8.1.1 Hardware Improvements

In order to prevent the on-board computer from collecting unnecessary frames and running the calculations on those frames, the rover only takes frames when the proximity sensor senses a potential object in the rover's direct path. Although the proximity sensor doesn't provide any information about the length, width or height of the object, ideally it tells the computer the current trajectory of the rover will cause a collision. When detected in time, the computer has enough time to calculate a new path. However, throughout the testing process, the proximity sensor used was highly inaccurate. Obstacles had to be artificially placed at specific positions in the rover's path to test the accuracy of the optical flow algorithm. A future implementation of the rover must have an accurate proximity sensor if the obstacle avoidance is to be done autonomously.

8.1.2 Software Improvements

Although an accurate proximity sensor can detect some object in the rover's path, edge cases exist where a sensor alone cannot see an object. To additionally cover these edge cases, a computer vision based algorithm may be developed to approximate the distance between the rover and anything in the frame by analyzing the optical flow vectors [4].

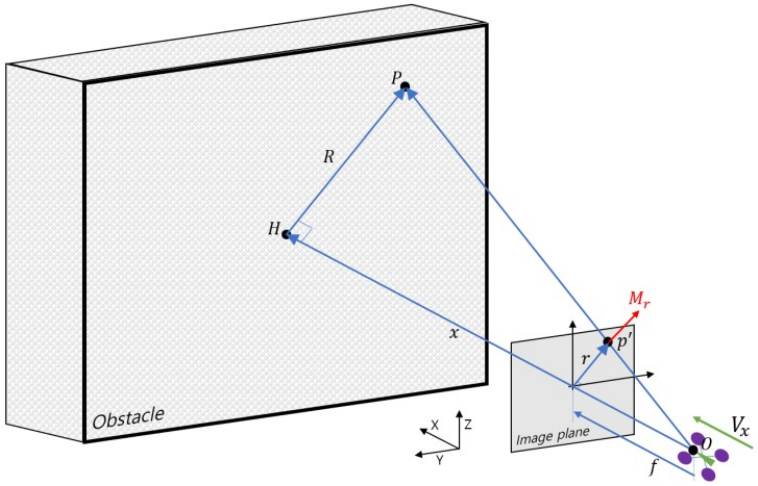


Figure 6: Optical Flow [4]

From Figure 4 we can derive an equation relating the optical flow vector as a function of the camera's focal length, the distance to the object, the vector R and the velocity:

$$M_r = \frac{fRV_x}{x^2} \tag{20}$$

The rover can use the proximity sensor alongside this distance measurement tool to more accurately decide if it should pursue a different path as a result of a possible collision. Another issue with

the software is the filtering of frames in varying environments (different lighting, solid colored walls, etc.). Since the rover was built to survey different environments, the on-board camera gets exposed to a variation of colors and shades. As a result of this, a generalized filtering algorithm must be developed. Different filtering techniques include blurring, dilating and eroding. For the optical flow algorithm to work accurately, the filtering sequence used must emphasize certain objects in the frame, while discarding others. As previously discussed, this filtering algorithm could ideally be a combination of many different filters to cover a wide range of scenarios.

8.2 Geo-location

The geo-location hardware and features on the rover need to be significantly improved to provide a solid positioning foundation. Once the right hardware is implemented onto the rover, the coordinate following algorithm is easier to implement. The next step would be to integrate the coordinate path following with the optical flow based obstacle avoidance.

8.3 Ditch Realization

The rover built is mainly aimed towards collecting vital sensor data on large monotone landscapes. In this case it would also be very useful to implement a solution to avoid any barrier preventing the rover from doing its job. Such barriers would include steep and rigid ditches, unexpected holes in the ground or any inconspicuous land features. It is not a trivial problem to solve with optical flow. One solution may be to analyze the optical flow in these particular occurrences and build an algorithm to detect such an event. Even a hybrid solution similar to the obstacle avoidance hybrid solution posed in the paper is plausible. Though this issue was not precisely addressed with optical flow, the intentions behind the GNSS integration was to provide a simple level of failure prevention since the path has to be drawn out by hand.

8.4 Closed Loop Fail Safe Power Supply and Scripts

With a full charge, the rover is able to run for approximately 15 minutes continuously without having any external power supply. This of course is a massive drawback since a rover of such tasks would have to be able to run significantly longer. This issue could be solved with newer batteries alongside the ability to recharge off of sunlight. Even with more reliable batteries the rovers electrical system would also need a fail safe power shutdown sequence to avoid powering off prematurely, thus damaging the control board. The only hardware needed for such implementation would be a volt meter of some sort. Once the volt meter realizes insufficient electric potential, it sends a signal to the Raspberry Pi, which in turn triggers a shutdown script. This would reduce the abrupt shutdowns and create a safer electrical environment for the on board hardware.

9 Conclusion

This paper aims to implement computer vision based optical flow computations along with geo-spatial navigation to create an autonomous land-surveyor system. The implementation is broken into two main components: the computer vision based optical flow calculations and the communication with the GNSS. The optical flow algorithms determine how the rover should steer when faced with an obstacle. The obstacle avoidance was tested with two different environments. Dense, chaotic environments breakdown the optical flow algorithm due to its inability to differentiate between essential and non-essential objects in the frame. Additionally, better quality hardware, namely

the proximity sensor, is required to make accurate decisions about when the rover should consider changing its trajectory. The geo-spatial navigation path following was subpar due to the inaccurate sensor data. With better hardware, the rover's path will be dictated by a combination of the obstacle avoidance and the GNSS-based navigational algorithms; as the rover detects a possible collision, the obstacle avoidance will dictate its motion. Once the obstacle is successfully avoided, the GNSS-navigational algorithm will take over the control.

10 References

[1] J.L. Barron et al. "Performance of optical flow techniques". In: *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1992, pp. 236–242. DOI: 10.1109/CVPR.1992.223269.

[2] T. Camus et al. "Real-time single-workstation obstacle avoidance using only wide-field flow divergence". In: *Proceedings of 13th International Conference on Pattern Recognition*. Vol. 3. 1996, 323–330 vol.3. DOI: 10.1109/ICPR.1996.546964.

[3] Raffaella Carloni et al. "Robot Vision: Obstacle-Avoidance Techniques for Unmanned Aerial Vehicles". In: *IEEE Robotics Automation Magazine* 20.4 (2013), pp. 22–31. DOI: 10.1109/MRA.2013.2283632.

[4] Gangik Cho, Jongyun Kim, and Hyondong Oh. "Vision-Based Obstacle Avoidance Strategies for MAVs Using Optical Flows in 3-D Textured Environments". In: *Sensors* 19 (June 2019), p. 2523. DOI: 10.3390/s19112523.

[5] *Electric DC Motor*. URL: %5Curl%7Bhttp://www.surpass-hobby.com/index.php?route=product%5C%2Fproduct&path=20_97&product_id=137%7D.

[6] Gunnar Farnebäck. *Two-Frame Motion Estimation Based on Polynomial Expansion*. URL: https://www.researchgate.net/publication/225138825_Two-Frame_Motion_Estimation_Based_on_Polynomial_Expansion. (accessed: 2022.03.02).

[7] David J. Fleet and Yair Weiss. "Optical Flow Estimation". In: *Handbook of Mathematical Models in Computer Vision*. Ed. by Yunmei Chen, Olivier Faugeras, and Nikos Paragios. Springer, 2006. Chap. 15, pp. 237–257.

[8] Warren Gay. *Raspberry Pi Hardware Reference*. Jan. 2014. ISBN: 978-1-4842-0800-7. DOI: 10.1007/978-1-4842-0799-4.

[9] Chris Harris and Mike Stephens. "A Combined Corner and Edge Detector. Alvey Vision Conference". In: 15 (1988).

[10] Berthold K.P. Horn and Brian G. Schunck. "Determining Optical Flow". In: *Techniques and Applications of Image Understanding*. Ed. by James J. Pearson. Vol. 0281. International Society for Optics and Photonics. SPIE, 1981, pp. 319–331. DOI: 10.1117/12.965761. URL: https://doi.org/10.1117/12.965761.

[11] *Ice Tower fan for pi*. URL: %5Curl%7Bhttps://www.waveshare.com/wiki/ICE_Tower_Fan_for_Pi%7D.

[12] Jeongdae Kim and Yongtae Do. "Moving Obstacle Avoidance of a Mobile Robot Using a Single Camera". In: *Procedia Engineering* 41 (2012). International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), pp. 911–916. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2012.07.262. URL: https://www.sciencedirect.com/science/article/pii/S1877705812026628.

[13] Bruce D. Lucas and Takeo Kanade. "International Joint Conference on Artificial Intelligence". In: *International Joint Conference on Artificial Intelligence*. 1981, pp. 674–679.

[14] Neeta Nemade and V. V. Gohokar. *Comparative Performance Analysis of Optical Flow Algorithms for Anomaly Detection*. URL: https://ssrn.com/abstract=3419775%20or%20http://dx.doi.org/10.2139/ssrn.3419775. (accessed: 2019.05.18).

[15] *Pigpio Library*. URL: %5Curl%7Bhttps://abyz.me.uk/rpi/pigpio/%7D.

[16] Nathaniel Pinckney. "Pulse-width modulation for microcontroller servo control". In: *Potentials, IEEE* 25 (Feb. 2006), pp. 27–29. DOI: 10.1109/MP.2006.1635026.

[17] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[18] *Sony Spresense Board*. URL: %5Curl%7Bhttps://developer.sony.com/develop/spresense/%7D.

[19] *Step-down DC-DC converter module for Raspberry Pi*. URL: %5Curl%7Bhttps://www.sunfounder.com/products/step-down-converter-module%7D.

[20] *The Savage X 4.6*. URL: %5Curl%7Bhttps://www.hpiracing.com/en/kit/109083%7D.

[21] Tomasi and Takeo Kanade. "Shape and motion from image streams: a factorization method." In: *Proceedings of the National Academy of Sciences* 90.21 (1993), pp. 9795–9802. DOI: 10.1073/pnas.90.21.9795. URL: https://www.pnas.org/doi/abs/10.1073/pnas.90.21.9795.

[22] Ilker Ünal and Mehmet Topakci. "Design of a Remote-Controlled and GPS-Guided Autonomous Robot for Precision Farming". In: *International Journal of Advanced Robotic Systems* 12.12 (2015), p. 194. DOI: 10.5772/62059. eprint: https://doi.org/10.5772/62059. URL: https://doi.org/10.5772/62059.

[23] Author Akshay Upadhyay. *Formula to find bearing or heading angle between two points: Latitude longitude*. May 2019. URL: https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/.

[24] Jessica Van Brummelen et al. "Autonomous vehicle perception: The technology of today and tomorrow". In: *Transportation Research Part C: Emerging Technologies* 89 (2018), pp. 384–406. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2018.02.012. URL: https://www.sciencedirect.com/science/article/pii/S0968090X18302134.

[25] Tianfan Xue et al. "The aperture problem for refractive motion". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3386–3394. DOI: 10.1109/CVPR.2015.7298960.