# 3D Passive Structured Light for Planetary Landers

**Major Project Report**

Spencer Newton

Institute of Mathematics and Physics

MPhys - Space Science and Robotics


May 2015

# Acknowledgements

I'd like to thank my supervisor, Dr Tony Cook, for helping me during various stages of the project where I was having difficulties, and to thank Dr Dave Langstaff and Steve Fearn for helping me set up the experiment. I would also like to thank my mentor, Alison Nash, for also providing support when I struggled to make progress, and for proof reading this dissertation throughout its development. Finally, I would like to thank my family and friends for supporting me through this project.

I would also like to thank the creators of OpenCV for providing a free open source library for computer vision tasks, and to the various internet users and tutorial creators who helped me understand computer vision and OpenCV along the way.

# Abstract

The 3-dimensional modelling of an environment is a crucial task of any robot where human interact is limited, such as on another planet. Many techniques are available, but few are reliable enough to be used on extra-terrestrial missions. One such technique is called Passive Structured Light, where sunlight reflected off of a mirror onto the surrounding environment is used to create a map of the area around a robot. The idea behind this technique is investigated in this report, focused more on the task of tracking the light spots in the environment using computer software, than the mathematics behind it.

The mathematical triangulation steps were attempted, but were not fully completed. Two algorithms to track the spots were tried and tested. One focused on isolating interesting areas in the image based upon colour, while the other did this based upon the difference between two successive images - one with a spot in it and one without. The experiment showed that both algorithms showed promise in simple situations, however when objects were added to the scene, different complications arose based on the type of object added. Further development and testing of the algorithms would be needed if they were to be used in the light spot tracking step of the technique.

# Contents

# 1 Introduction

A common problem that many robots face is the task of sensing and understanding their environment. Therefore, the ability for a robot to be able to create a 3D model of its surroundings is crucial. Robotic space missions often need to be able to create 3D models of their environment automatically in order to complete their tasks without aid from humans who will not be present. For this particular problem, many different techniques have been tried and tested, however only a few are applicable to space-based robotics.

There are many reasons for the lack of applicability of techniques to space robots. However it is mainly due to the limited resources available on planetary landers and rovers. Hardware performance, power consumption, and space availability are all important factors when it comes to deciding which 3D modelling techniques are used on these missions. Other factors, such as cost (as a result of weight on launch) and reliability must also be taken into account, and so investigations into new techniques to improve these problematic areas are worthwhile.

The most common technique so far for extra-terrestrial robotics has been the use of stereo vision, whereby two cameras separated by a short distance take images of the environment simultaneously and compare them for disparities between common points in both. Other techniques, such as Laser Range Finding and Structured Lighting (see Section 2.3), have been tried and tested on Earth, but have yet to make an appearance on major planetary robotics.

This report is focused on the investigation of a technique called Passive Structured Light (PSL) which is an adaptation of more common active structured light methods. In the technique of structured light, a single camera is used alongside a projector that is projecting a known pattern of lights (e.g. laser dots) onto an environment. The camera takes an image that contains the projected pattern, which will be distorted due to the shape of the environment. Based on the distortion / displacement of the pattern and the position of the camera and projector, the positions of the detected light spots on the environment can be derived.

The Passive version of this technique uses light (from the sun) reflected from a mirror to create a spot of light on the environment, which can then be detected. This is advantageous over other techniques because it would use less power, weigh less (and thus cost less), and could be used for scientific purposes other than 3D modelling (e.g. heating soil / rocks). This experiment provides a brief view of the maths behind it, but specifically focuses on the task of detecting spots of light that have been reflected off a mirror, which is a key part of the technique.

## 1.1 Passive Structured Light

The maths involved in 3D modelling using PSL is vector / triangulation based. A brief description of the steps that one would take to create a 3D model is summarized below, with the steps focused on in this report outlined in bold.

1. A robot with a camera and a small mirror (or set of mirrors) is stationary in an environment. The parameters of the mirror and camera should be measured and calibrated (position, angle, lens distortion etc) to be used in the coming calculations.
2. Light from the sun is incident on the mirror, which is reflected and creates a spot of light on the environment. The location of the spot depends on the position of the sun and the position & angle of the mirror.

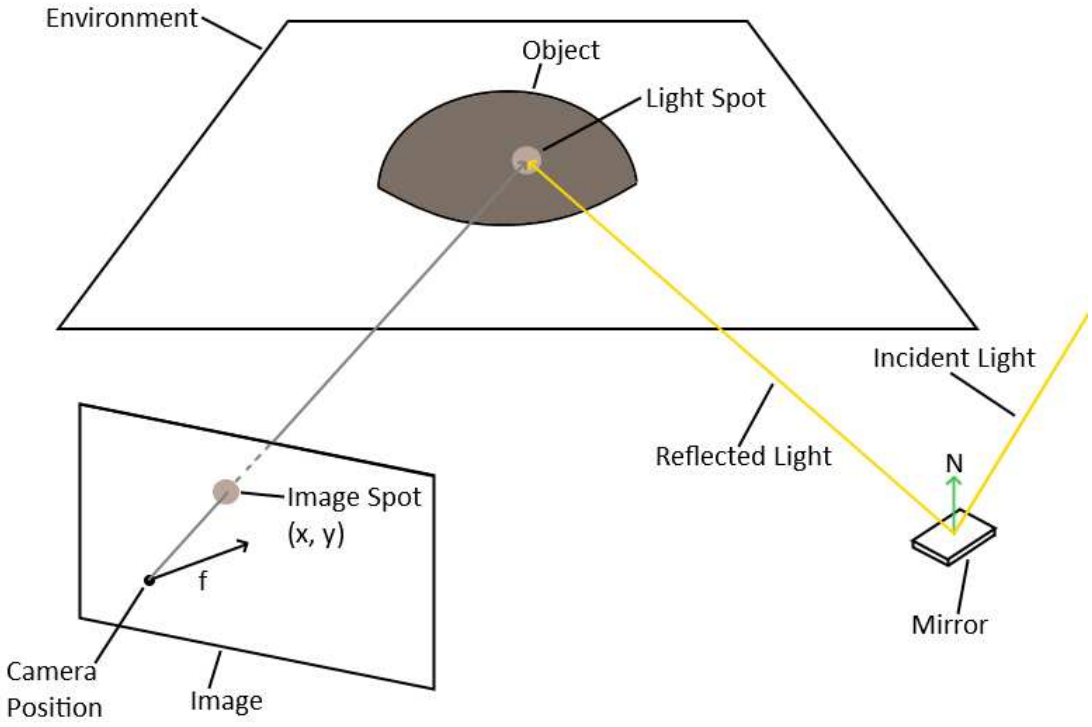



Figure 1: An illustration showing a basic PSL equipment setup.

3. The camera then takes an image of the environment (with the spot in view).

4. **Using the image from the camera, the position of the spot in the 2D image should be discovered.**

5. **From this information, two vectors can be derived (assuming that there is only one spot - multiple pairs of vectors would be derived for multiple spots). The first is the vector of reflected light from the sun. The second is the vector representing the position of the spot as seen in the image from the point of view of the camera.**

6. These two vectors should intersect, or come very close to intersecting, as they both pass through the spot in the scene. By finding where they intersect we can discover the 3D location of the spot in the environment relative to the robot.

7. As the sun moves across the sky (or by moving the mirror), the spot moves along the environment as the light reflects at a different angle. By taking multiple images throughout the day (or by changing the angle of the mirror), many points of intersection can be found and thus a 3D 'point cloud' can be created which represents the surrounding environment.

The position of the sun is measured in Right Ascension / Declination, which should be converted to Altitude / Azimuth in order to get a vector of the light incident on the mirror. This light vector can then be reflected in the mirror (or mirrors) with the equation:

$$R = 2(N \cdot L)N - L \tag{1}$$

where $R$ is the reflected light vector, $L$ is the incident light vector, and $N$ is the normal vector of the mirror face.

After taking an image with the camera, the spot should then be found in it and the pixel coordinates of the spot should be recorded. Using this, a vector from the position of the camera 'through' the spot in the image can be calculated. These two vectors can then be triangulated to find the closest point of intersection.

For the triangulation, it is best to represent the vectors as lines in parametric form, such that:

$$P = P_1 + u(P_2 - P_1) \tag{2}$$

where $P$ is the end point of the line, $P_1$ is the origin point, $P_2$ is another arbitrary point on the vector, and $u$ is a scalar quantity that represents the length of the line. If both vectors are represented in this way, that two values of $u$ can be incremented slowly until the end points are the same (or as close as they can get). This closest point represent the position of the light spot relative to the camera / mirror.

The maths behind the process was briefly tested, however the main focus was on the task of tracking the spot in the image. The next section details the software used to do this.

### 1.2 Light Spot Tracking

While the task of finding light spots in an image could be done by a human, in a real application of this technique the task would preferably be done automatically by the robot. Therefore, certain computer vision techniques need to be applied to the images in order to find the light spots. For the image processing done in this report, a software library called OpenCV (Open Source Computer Vision [1]) was used, which contains many functions that allow for the manipulation and processing of images. A program was written (based on tutorials by Kyle Hounslow [2]) to track the light spots using OpenCV and common object-tracking techniques. The work from [2] was focused on object tracking methods via video, but was modified to take a selection of images as input rather than video. A view of the resulting program can be seen in Figure 9 and the source code for the program can be found in the Appendix, Section A.

Two algorithms, used in conjunction with features from OpenCV, were experimented with using images gained from the experiment. One is based on isolating certain colours from the image, which assumes that the light spots will be of a particular colour (including white). The other is based on comparing the image with the spot against another image of the same view without the spot, and looking for large variations in the pixel values. An area with a large difference is indicative of there being a bright spot on the surface.

Both of these algorithms were tested independently of one another and the method by which they were tested is outlined in Section 3. The results of the experiments can be found in Section 4, and a discussion of the results can be found in Section 5.

# 2 Literature Review

**Note: This literature review has been included for completeness, and was written before the full report. Therefore some parts may appear repeated from the introduction and theory sections.**

## 2.1 Introduction

Most planetary missions that include ground-based robots must have a reliable method for modelling their surroundings in a way they can understand. This is vital for many scientific and navigation tasks, as the location of objects relative to the rover is needed to be known in order to experiment on or to avoid them. While having a camera take images of the environment around the robot is good, it does not provide the three-dimensional data that is needed for complex tasks, such as manoeuvring a robotic arm to take samples from an interesting rock. 3D models are useful for the rover's own planning, but are also very useful for scientists who wish to examine the environment and plan the mission themselves.

Many different methods of getting 3D information from an environment have been developed for Earth-based robotics, however it is difficult to apply them to space and planetary robotics. This is because the hardware on the robots we send to other planets is very limited in comparison to Earth hardware. Hardware is restricted by space, power, and weight due to the high cost per kg incurred by sending them to other planets. Therefore it is important that the techniques we use to map environments save resources in these areas.

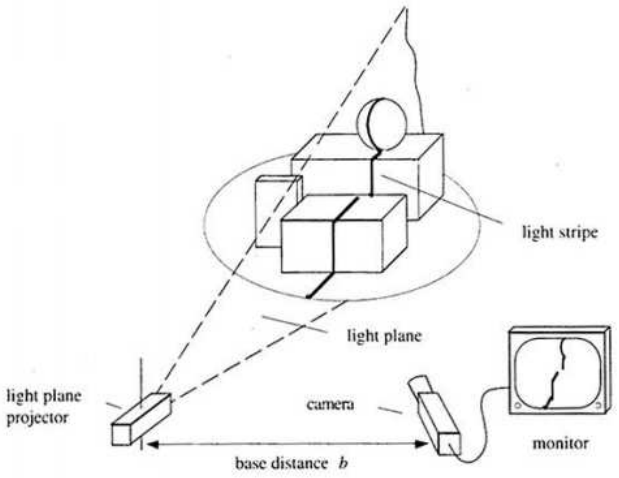



Figure 2: An illustration showing the use of active structured light to determine the structure of an environment. Source: `http://www.sci.utah.edu/~gerig/CS6320-S2012/Materials/CS6320-CV-S2012-StructuredLight.pdf`

The aim of this project is to investigate the possible use of a technique called Passive Structured Light (PSL) to recreate 3D models for planetary landers. PSL is an adaptation of a technique called Structured Light (SL) where an image of a scene is taken with a camera after a light pattern has been projected onto it. 3D range information for each pixel can be obtained by comparing the projected pattern with the camera image. The passive version of this involves using an array of mirrors to reflect natural light onto the scene as a replacement for the projector. The feasibility of this approach will be tested, in areas such as calibration and range, and an attempt to model an environment using range data will be made.

## 2.2 Background Physics

Structured light is where a pattern of light (which varies on application [3] [4]) is projected onto a scene and a single camera (or multiple [5] [6]) is used to capture the resulting scene (Figure 2). The pattern can be viewed as multiple parallel lines (in the case of a stripe pattern) being projected into the environment, where each line can be represented as a geometric plane. The planes make contact with the surface of the scene and the resulting lines can be seen and captured

by the camera. Using computer vision techniques, the lines on the surface can be detected in the pixels of the image. For each resulting pixel, a ray can be created that points 'through the image' to the contact point of the surface and the plane. The point at which the ray and plane intersect (or come closest to intersecting) is the surface point, the coordinates of which can be calculated (relative to the camera).

Passive Structured Light takes this a step further by removing the projector and replacing it with an array of mirrors. The mirrors are set up so that they reflect sunlight onto the scene, creating discernible spots of light. As the sun moves across the sky, the 'sunspots' trace paths over the terrain, and by taking multiple images over time, the paths can be recreated. These paths are essentially a stripe projection pattern (that would have been shone onto the environment using the projector), the planes of which can be realised with reflection geometry and information about the angle of the mirror and the position of the sun in the sky. 3D range information can then be derived in the same way as for regular Structured Light.

## 2.3 Literature

There have been many different successful applications of 3D modelling techniques in the world of robotics. However relatively few end up being used for the purposes of space robotics due to the limitations mentioned previously. This section will look at other 3D modelling techniques, their applications to general robotics, and also to the specific application of space and planetary robotics.

### 2.3.1 3D Modelling Techniques

A common technique for 3D modelling is called Stereo Vision. This is where the disparity of corresponding features between two camera images are used to obtain depth information (see Figure 3). This technique has been widely used ([7] [8]), however it has it's problems. The most prominent problem is correspondence - how two corresponding features are found in both images. This is somewhat easily done with computer vision and artificial intelligence, but these do not help when the environment is very plain (ie there are no distinguishable features) or in extreme lighting conditions. Although this is a problem for passive stereo vision, active stereo vision could be used to project discernible features onto a scene ([6] [5]) similar to the way structured light would. Shadows could also be used to add features to a scene ([9]).

Another problem of stereo vision is the range at which it can be used, which is only for a few meters away, after which the disparities are negligible. This can be improved by increasing the baseline distance of the cameras. It is possible to have a variable baseline with only one camera by taking two images from different positions by moving the robot, but this is not possible with certain robotics (such as a planetary lander). This is called Camera Motion Stereo [10] [11], or alternatively Zoom from Stereo if the robot moves towards the scene, creating a zoom effect. This suffers from similar problems regarding correspondence, however.

One final technique relating to the stereo theme is Virtual Multi-Camera Stereo (VMCS, [10] & [12]). VMCS uses one camera with a mirror attached to a robotic arm. The mirror is moved around to get multiple views of the environment with the camera. The resulting images from the camera contain an image of the scene, and a reflected image of the scene in the mirror. The image in the mirror can be used, after correction to account for reflection, as a second image from a 'virtual camera' in a stereo vision algorithm. This uses similar apparatus to the PSL technique. However

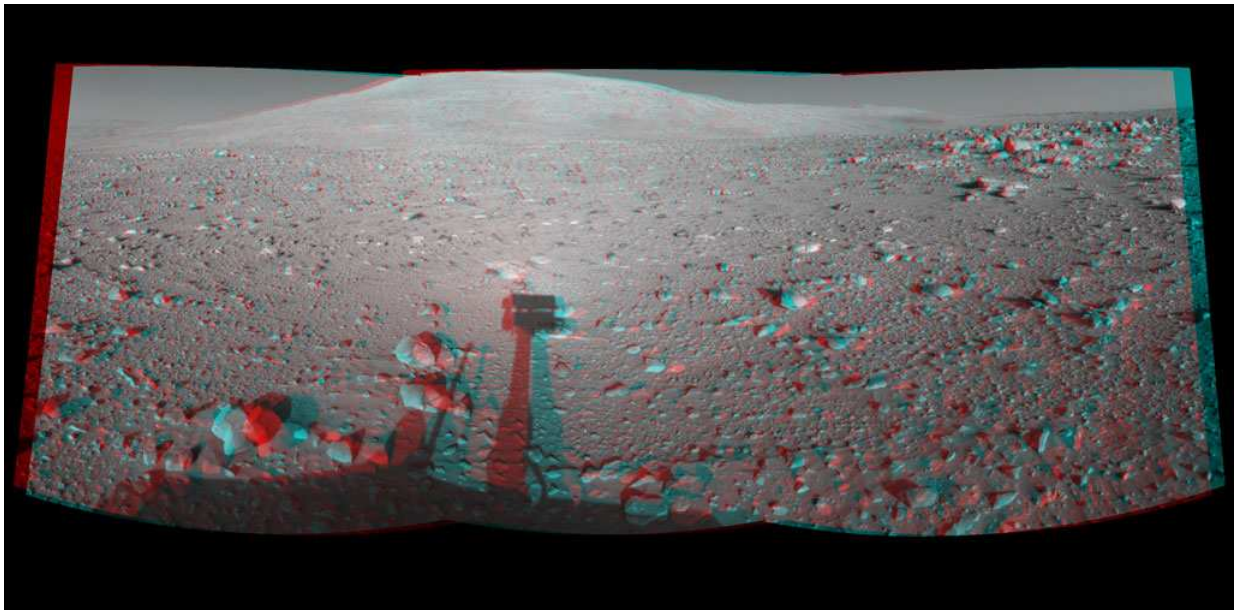


Figure 3: An anaglyph image showing the images taken by the stereo cameras of the Spirit MER. The blue & red sections are the disparities between the two images used to determine depth information. Source: `http://mars.nasa.gov/mer/gallery/images.html`

if the mirror is small then the VMCS process will have to be repeated multiple times to get a full model of the environment.

A technique that is popular with many earth-based robots is laser range-finding (LRF). This has been used in many areas, including indoor and outdoor environment mapping ([13] [14]) on Earth. This technique involves emitting a focused laser beam towards an object and detecting the reflection. The time taken for the beam to travel to & from the object at the speed of light can be used to calculate the distance of the point of reflection. Scanning a scene with lots of lasers in a $360^o$ angle around a robot can generate dense point clouds to model the environment.

Laser range-finding has many advantages and disadvantages compared to other techniques. LRF is an active method, and so can be used at night, has a much longer range than other techniques, and can generate very dense point-clouds. It heavily relies on the fact that the target surface is reflective, which it may not be, and thus could be totally useless in certain situations. Since the calculations rely on the speed of light, which is very large, the resolution can be poor at certain ranges as it may not be possible to measure small enough time differences. Atmospheric distortions may also alter the path travelled by the laser and thus give inaccurate results.

Finally, and most relevant, Structured Light is a promising technique for modelling environments. It has been used in many different applications before, such as for terrain surface classification ([15]) and general object modelling ([16], [17]). SL can be used in many variations ([3]) to create 3D models of large and small environments ([6], [5], [15], [16], [17]), on static or dynamic objects ([4]) at various degrees of accuracy and resolution. Alternative methods that do not require projectors include using shadows to cast patterns across a scene in order to create features ([18]), and of course PSL.

An advantage of SL is the varied apparatus compared to other techniques. The use of both

projectors and cameras allows us to capture regular images of an environment but also project light onto the scene, which could be used for various scientific purposes (ie testing the reflectivity of a surface) and allows it to be used in dark areas. This projection also means that the terrain does not need to have points of interest in it for any feature correspondence algorithms to function. Certain SL techniques do not work very well with dynamic scenes, however other versions can handle it at the cost of lower accuracy ([4]). This would not be a problem on another planet where scenes will most likely be static.

### 2.3.2 3D Modelling in Planetary Robotics

When it comes to applying the best techniques to planetary robotics, many other factors are taken into account that would not be considered on Earth. These robots have to be sent to other planets, which is very expensive. The cost is proportional to the weight and so it is important to keep the weight low. The robot will have limited power, so it needs to be energy efficient, and it should not be too bulky so as to not take up too much payload space that could be used for other instruments. In addition to this, it also has to function correctly in an environment very different to that of Earth. Capabilities such as thermal limitations, radiation and space-travel tolerance, and any other environment-specific properties need to be considered.

For planetary robots, it is also very beneficial to have an instrument capable of performing many tasks, due to the limited space. With these camera-based techniques, additional uses can be derived such as using the cameras to detect wavelengths other than visible light. This is useful for, among other things, classifying materials on other planets without having to physically gain samples. For the laser or projection based techniques, additional use can be gained as well, such as by testing the reflectivity of a particular surface.

For previous planetary missions there is not a lot of diversity. The most common technique is Stereo Vision, which most rovers (*Spirit*, *Opportunity*, *Curiosity*, *Chang'e* etc) and landers (*Phoenix*, *Pathfinder*, *Chang'e* etc) have been equipped with thus far ([7] [19] [20] [21] [22]). This is because the instruments are reliable and use relatively little power, but still provide good quality models and images for the static environments of other planets. Since the cameras are fixed in place they don't have any mechanically moving parts, which could malfunction.

The Mars Exploration Rovers *Spirit* and *Opportunity* both used Stereo Camera Vision over other techniques. Spirit landed in a crater with plenty of rocks (features), whereas Opportunity landed in a flat plain with mostly featureless soil (Figure 4). The stereo matching algorithms worked well for Spirit, where there were lots of obvious features to track, but Opportunity had to use the higher resolution navigation cameras (instead of the hazard cameras) to detect smaller features in the soil to get adequate range data ([23]). The hazard cameras used stereo vision to create 3d models in order to avoid collisions.

The *Phoenix* lander utilized Stereo Vision to visualize the terrain around itself in the Martian polar regions. The camera was 1024 x 1024, could see in multi-spectral images at 12 wavelengths for analysis. Data was used to create Digital Elevation Models for the robot arm, which also had a camera on it. The robot arm was used to conduct scientific experiments, and could not have done so without a 3D model of the environment ([19]). Aside from NASA Mars projects, Lunar missions utilized Stereo cameras for imaging as well. The Chinese *Chang'e* mission deployed to the moon consisted of a Lander and a Rover, both of which used stereo cameras ([20]). This shows that stereo vision can work well in many environments, so long as the environment is sufficiently textured.

Figure 4: Images showing views from the areas around the landing zones of MERs Spirit and Opportunity. The lower image (Opportunity - Meridiana Planum) shows a lack of major features, while the top image (Spirit - Gusev Crater) has an abundance of rocks that can be used for the stereo matching algorithms. Source: `http://mars.nasa.gov/mer/gallery/images.html`

While laser range-finding has been used extensively on Earth, the disadvantages of it factor heavily into it's lack of use for planetary robots. The equipment required to use this technique necessitates the mechanical scanning of the environment, and this equipment could malfunction or break during transit. The most outstanding disadvantages are the large, heavy and power-hungry instruments used in the process. In addition to this there are no cameras to take real images, which are needed for other purposes, and the surface of different planets (ie Mars) may be very dusty, which can interfere with the required scattering of the laser pulses. This and other factors outweigh the benefits laser range-finding would have ([24]).

However, laser range-finding for use in planetary robotics has been experimented with ([25]). During the 2008 ESA Lunar Robotics Challenge, a lander with two accompanying rovers was tested in a mock lunar mission, whereby the goal was to explore a crater in constant darkness (a possible scenario on the moon). Stereo cameras could obviously not be used in this situation, and so laser

range-finding was used to map the dark crater, the data from which was relayed back to the lander. While not specific to small-scale planetary surface modelling, the MOLA satellite orbiting Mars uses a laser altimeter to create large-scale digital terrain models.

Despite the prevalence of stereo cameras and disadvantages of laser range-finding, the Mars Pathfinder rover Sojourner used a simple Structured Light technique that combined these two techniques to scan it's surroundings ([21]). It utilized a simple laser stripe scanner and single camera to detect and avoid hazards during it's autonomous operation. This was used in 1997, which shows that structured light can be used efficiently and at low power consumption for simple environment modelling and hazard avoidance ([24] [26]).

### 2.3.3 Passive Structured Light

The idea of Passive Structured Light is a relatively new adaptation of the regular Active Structured Light method. As such, there are no such published papers describing previous in-depth attempts. Two articles ([27] & [10]) discuss the possibility of using Passive Structured Light instead of other 3D modelling techniques and briefly assess the feasibility, but they do not go into tremendous detail.

The possibility of using PSL is appealing due to the advantages it would bring to a mission. The second camera from a stereo set up, or the projector from a structured light set up, would be replaced with a set of movable mirrors. This would mean less electronics and less power would be consumed by the instruments, in addition to it weighing less. The sunspots can also be used to illuminate shadowed areas, and could be distinguished from the surrounding terrain for up to 20m away from the source ([27]). They could also be used for various spin-off science purposes, such as using them to perform photometry on a particular surface, or perhaps using them to heat a soil sample.

PSL does have disadvantages and problem areas (such as it being totally reliant on the sun), but it would still be worthwhile to experiment on its effectiveness in the area of planetary robotics.

Figure 5: A photo showing a set of light spots, reflected off of a set of mirrors, being tracked over an object. The top image shows one image taken of the spots. The bottom image shows the spots imaged at 2 minute intervals [27].

# 3 Experiment

The experiment was split up into two parts. The first experiment was focused on getting accurate measurements in order to perform the vector calculations for the theory, and the second was to gather images of light spots in different situations for the light spot tracking software. The gathering of images did not need precise measurements and so the precise arrangement of equipment was less important.

## 3.1 Apparatus

While in a real application of this technique light from the sun would be used, due to inconsistent weather conditions the experiment was conducted inside. To replace the sun, an LED torch was used as a light source and ambient light was also present from lighting in the room. The surface the light spot was aimed at was a piece of white grid paper on a flat desk surface, which had $x$ & $y$ axes drawn onto it. In later experiments three objects were place into the scene. One object was a triangular prism made of paper, another was a black plastic laptop charger, and the last was a rubiks cube.

A mirror of an appropriate type (small and front-luminized) was not available, and so two highly reflective metal discs (henceforth referred to as the 'mirrors') were used to reflect light. These were not the right shape to create small light spots, and so a 'mask' was cut out of card and placed over the reflective surface in order to get certain shapes and patterns of spots. The three masks used were made to have one hole (1cm * 1cm), nine holes (1cm * 1cm each in a 3*3 grid), and thirty-six holes (0.5cm * 0.5cm each in a 6*6 grid) open to allow light to reflect through them. To keep the mirror at a fixed position and angle, a stand and clamp was used. For the second experiment a stand and clamp was used to hold the camera steady. The layout of the equipment can be seen in Figures 6 & 7 for experiments 1 and 2 respectively

## 3.2 Method

### 3.2.1 Experiment 1

The first experiment was to gather data to test the geometry theory. The apparatus was set up as shown in Figure 6 with the light source pointed directly at the mirror, causing a light spot to appear on the grid paper. The axes on the paper were used to record the position of the spot, mirror (the position of the mask hole), and light source in centimeters relative to the origin in the bottom right hand corner. A ruler was used to measure the height of the mirror & light source. Once this data had been gathered, an object (in this case the rubiks cube) was placed in the path of the reflected light, which caused the spot to appear on one of the cube faces. The position of this re-positioned light spot was then recorded as before.

### 3.2.2 Experiment 2

The second experiment was to gather images of different situations to test the spot tracking software. For this, seven different situations were tested, where the objects in the scene, lighting conditions, and number of spots were changed. The enumeration shown here corresponds to the test numbers (the first situation listed here is test 1, etc) used later on in the report. The different tests were:

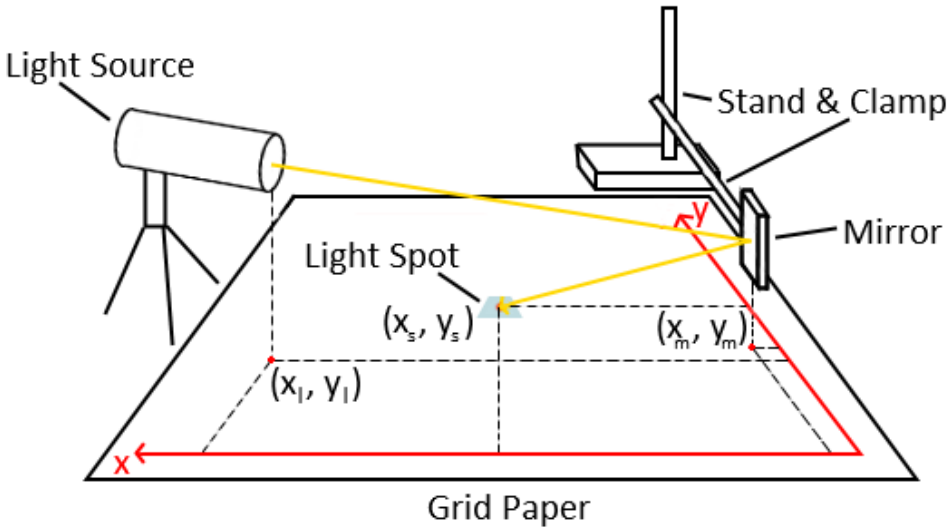



Figure 6: An illustration showing the experimental setup for the first experiment.

1. A single spot, with no ambient light, and no object in the scene.
2. A single spot, with ambient light, and no object in the scene.
3. Multiple spots (9), with ambient light, and no object in the scene.
4. Multiple spots (9), with ambient light, and a black plastic object in the scene.
5. Multiple spots (9), with no ambient light, and a multi-colour rubiks cube in the scene.
6. Multiple spots (36), with ambient light, and no object in the scene.
7. A single spot, with ambient light, with a small triangular prism made out of the same paper in the scene.

For all of these tests, the apparatus was set up, with the camera looking at the scene in a stationary position as shown in Figure 7. An image was taken of the scene without a light spot present (for a future image processing step), then the mirror was moved / angled, in no precise way, to move a spot of light over the scene. At various (six) points during the spot motion, an image was taken with the camera. This was repeated for each test.

### 3.3 Image Processing

Once the images of the different situations had been taken, they needed to be processed to discover any light spots present in them. This was done using two different algorithms to find areas of interest in the images. The methods of each algorithm is explained in the following sections.

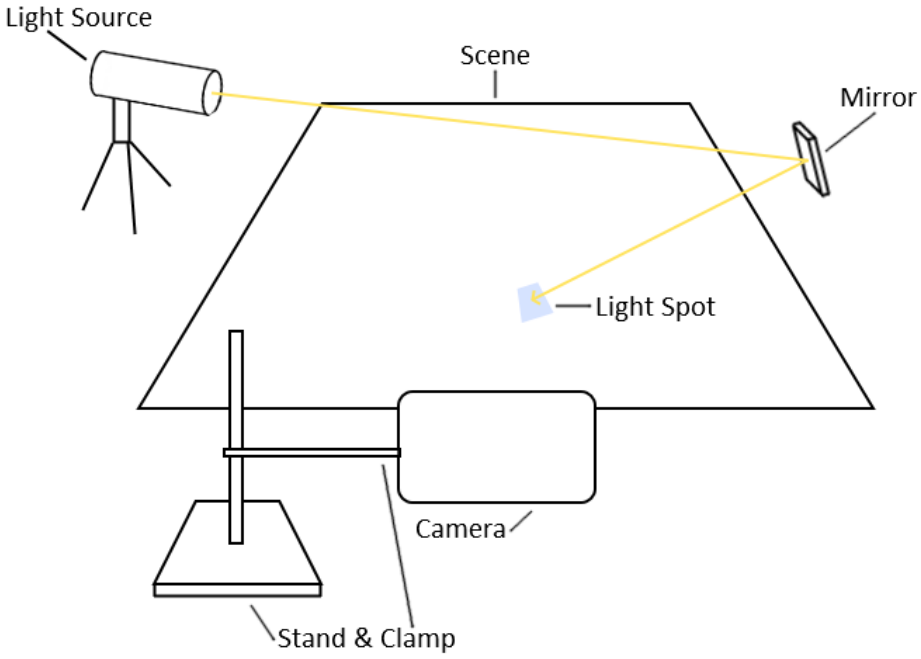



Figure 7: An illustration showing the experimental setup for the second experiment.

### 3.3.1 Colour Tracking Algorithm

First, the image is loaded into the program as a $w$ * $h$ * 3 matrix of pixel values, where $w$ and $h$ is the width and height of the image, and the three 'layers' represent the blue / green / red values of the pixels. This image is then copied to another matrix and converted to the HSV colour space in the process. The HSV system (see Figure 8) does not represent colours as the relative intensities of red / green / blue, but rather represents them as a hue (the colour), a saturation (how 'full' the colour is), and a value (how bright the colour is).

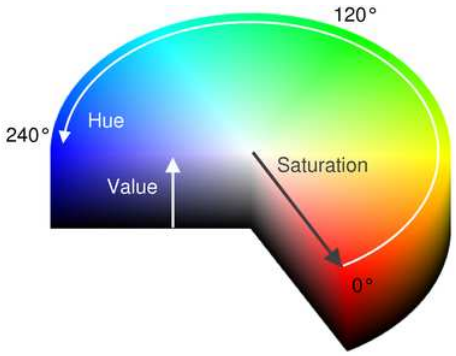



Figure 8: An image showing the idea behind representing colours in HSV (Hue-Saturation-Value) format.

This HSV image is then copied again to a 'threshold' matrix, and has a binary thresholding operation applied to it (Equation 3). This operation causes the resulting image to be populated with pixel values of either 0 or 1. If the colour value of a pixel in the HSV image is between a set of parameters (the threshold), then it is a 1 in the threshold matrix, otherwise it is 0. In this case, the threshold is a set of maximum and minimum values for the hue, saturation, and value parameters. The software allowed these parameters to be changed, and so they were altered to values that could isolate the light spots.

$$result(x,y) = \begin{cases} 1 & \text{if min(hsv)} < \text{src(x,y)} < \text{max(hsv)} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

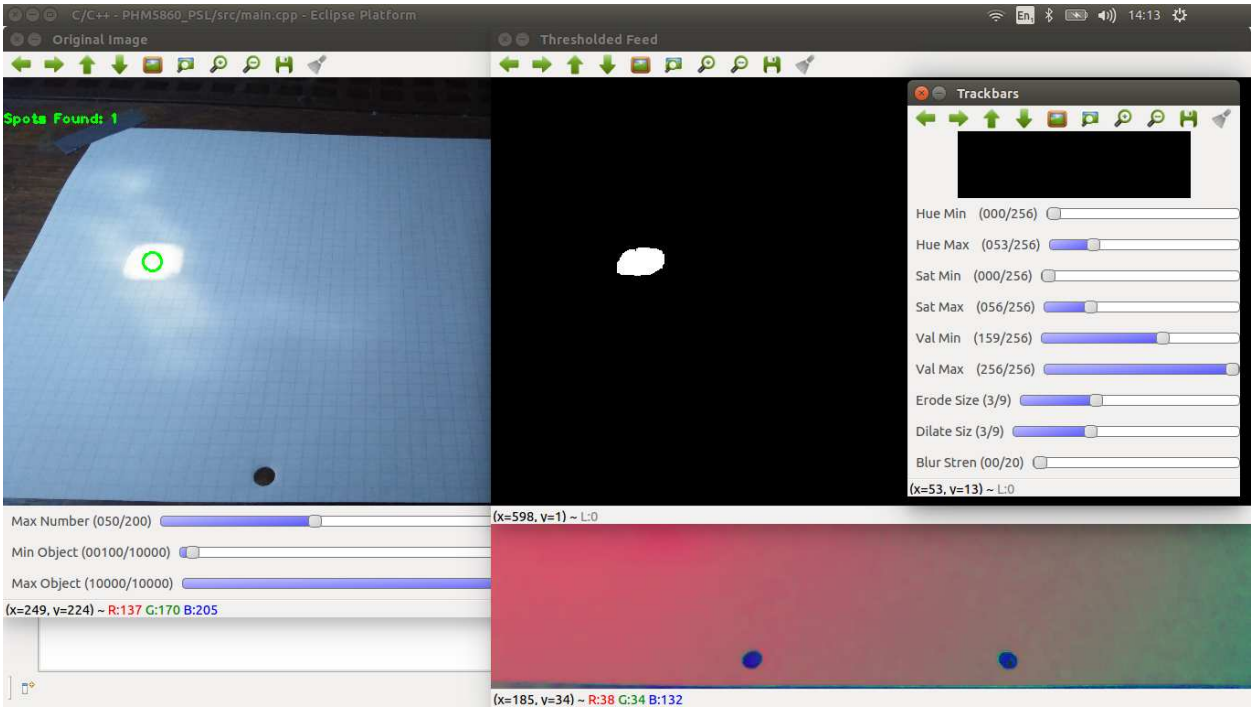



Figure 9: A screenshot showing the display screens of the program. Four windows are visible: The small vertical window on the right contains track bars that allow the user to change the threshold parameters, the window behind that is the resulting threshold image, the window behind that and to the left is the raw image with any light spots marked, and out of view (the pink window) was a window showing the raw image converted to the HSV colour space (unused in this report).

### 3.3.2 Difference Tracking Algorithm

First, the spot image is loaded into the program as a $w * h * 3$ matrix of pixel values, like before, but this time the image of the scene without a light spot in it was loaded as well. These two images are then copied to new matrices and are converted to gray scale in the process. Each corresponding pixels of the gray scale images is then compared to one another, and the absolute difference between the values is recorded in another matrix. This difference matrix is a gray scale image showing the difference between the two images.

This image is then thresholded like before, but instead of using minimum and maximum HSV values, a minimum difference value (the sensitivity) is used to threshold the image (Equation 4). Areas where the difference is large enough for us to suspect there is a spot will be white, and the sensitivity value could be altered to help find the spots.

$$result(x,y) = \begin{cases} 1 & \text{if sensitivity} < \text{src(x,y)} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

### 3.3.3 Post-Processing

After these processes, there was often background noise where extra reflections or anomalous readings were picked up. To eliminate these, and to make the spots more pronounced, the threshold images were put through morphological operations known as erosion and dilation. These processes effectively removed any small areas of high intensity (erosion) and then made any remaining high intensity areas larger (dilation). This process of erosion and then dilation is called 'opening' an image.

Both operations are done by scanning each pixel in the threshold image and looking at any surrounding pixels. In erosion, each pixel is compared to the surrounding pixels (in a square 'kernel' around it), and is changed to the lowest value surrounding it. Dilation is the opposite, each pixel is set to the highest pixel value surrounding it. The size of the kernel can be changed to make the effects bigger or smaller.

The resulting threshold image should contain distinct white areas without small blobs of background noise. Using functions provided by OpenCV, information about these areas, such as number of blobs, contours, shape, area, and most importantly coordinates in the image can be derived. These coordinates were used to draw circles around spots found in the original image, and to count how many were found.

The parameters required to find spots between different tests varied. Preferably, the software should be able to track all the spots throughout the images with a single setting, and so the parameters were initially set to find the spots in the first image of each set. These parameters were tested for each consecutive image so see how well they continued to track the spots. If they were unsuccessful, the parameters were altered in an attempt to find the perfect settings.

The result from the software would be coordinates for the positions of the light spots in the images. However in this case it was important to check that the algorithm had successfully identified the spots, and so images showing the results of the two methods were gathered. These are shown in the next sections.

# 4 Results

This section displays the results gained from the two experiments outlined above. The geometry results section shows the results of experiment one to test the theory, while the software results sections displays the results of the light spot tracking experiment and software.

## 4.1 Geometry Results

Below is a table showing the measurements and calculation results from the first experiment (shown in Figure 6) based on geometry. The calculations were programmed into an excel spreadsheet. The first table contains the measurements made during the experiment, and the second & third tables show the calculations made using them.

| | Mirror Position | Light Source Position | Spot (Surface) Position | Spot (Cube) Position |
|---|---|---|---|---|
| x | 2.20 | 23.40 | 17.50 | 10.30 |
| y | 9.20 | 11.70 | 7.80 | 8.50 |
| z | 5.80 | 13.00 | 0.00 | 3.00 |

Table 1: A table showing the measurements made during experiment 1.

| | Light Vector | Mirror Normal Vector | Reflected Light Vector | Reflected Light Vector Normalized |
|---|---|---|---|---|
| x | -21.20 | 1.00 | 21.20 | 0.94 |
| y | -2.50 | 0.00 | -2.50 | -0.11 |
| z | -7.20 | 0.00 | -7.20 | -0.36 |

Table 2: A table showing the calculated vectors of the light source to the mirror (Light Vector), the light vector reflected in the mirror normal (Reflected Light Vector), and the normalized version of the reflected light vector (of length 1.00).

| | Reflection Vector (Surface) | Reflection Vector (Cube) | Surface Vector Normalized | Cube Vector Normalized |
|---|---|---|---|---|
| x | 15.30 | 8.10 | 0.93 | 0.94 |
| y | -1.40 | -0.70 | -0.09 | -0.08 |
| z | -5.80 | -2.80 | -0.35 | -0.32 |

Table 3: A table showing the calculated vectors of the light spots, from mirror to the flat surface & from mirror to cube face, and the normalized version of these vectors (of length 1.00).

## 4.2 Software Results

Below is a list of labelled images from the second experiment (shown in Figure 7) after the image processing steps. To save space in this report, instead of showing the raw images from the camera, the software-edited camera images are displayed since the only additions are the green circles over where spots were found and a spot counter. While many images were taken, only a selection of them

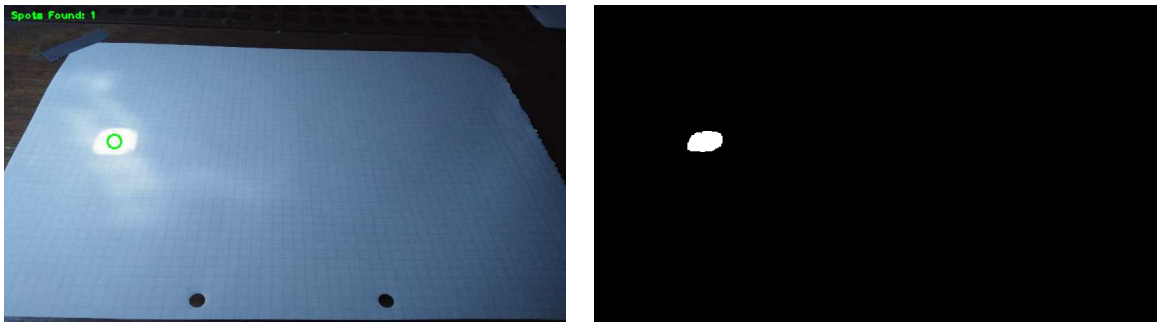
are shown to save space, and the selected images all display something discussed in the analysis section.

The results below show 4 different types of images. The two common types are the black and white 'threshold' images and the raw images the software alters to show spots with green circles around them. The other two images are used in the difference tracking algorithm. One is a 'base' image, taken before each test, that does not include any light spots. The other is a gray scale image showing the intensity difference between pixels in the base image and spot-image. These are captioned as 'Base Image' and 'Pixel Intensity Difference' respectively.

The pairs of threshold / tracking images that are side-by-side are from the same settings, and the tracking images have captions showing the parameters used in the test. The HSV parameters are contained within square brackets (e.g. [a-b, c-d, e-f]), where 'a-b' is the minimum-maximum hue values, 'c-d' is the minimum-maximum saturation values, and 'e-f' is the minimum-maximum value possibilities. Some Figures are continued over multiple pages.

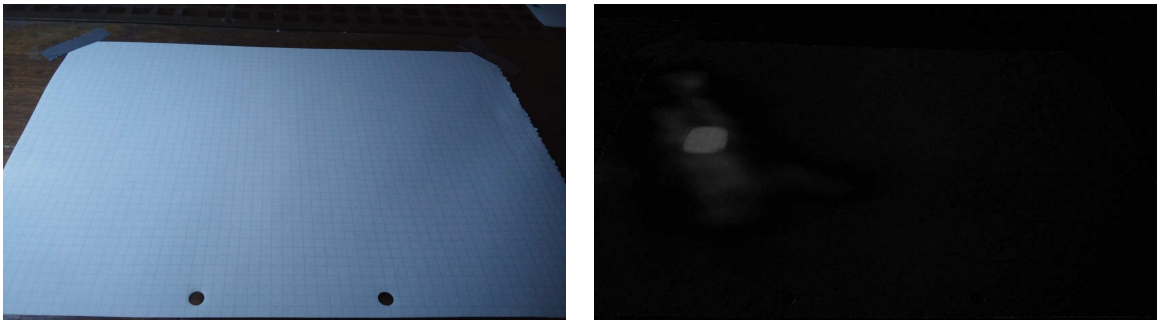
### 4.2.1 Test 1

This test was done without ambient light and with a single bright spot reflected off of a mirror.

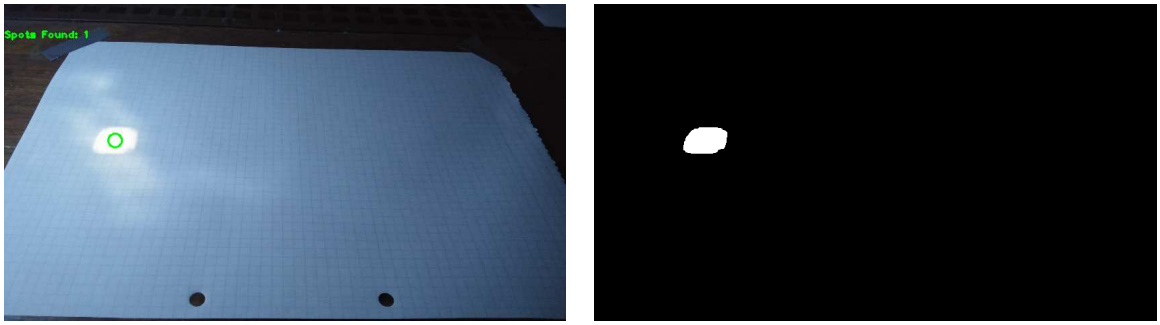



(a) HSV = [0-179, 0-10, 245-255] (b) After Threshold.

Figure 10: Software results of the colour tracking algorithm for test 1.



(a) Base image (b) Pixel Intensity Difference

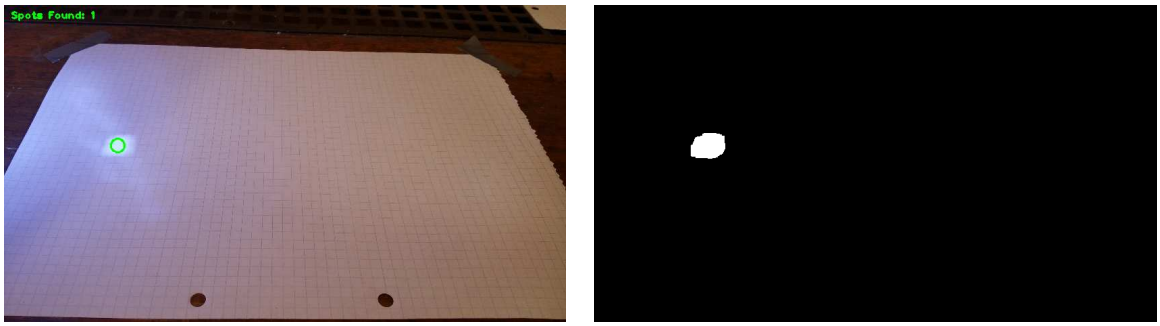



(c) Sensitivity = 40. (d) After Threshold.

Figure 11: Software results of the difference tracking algorithm for test 1.

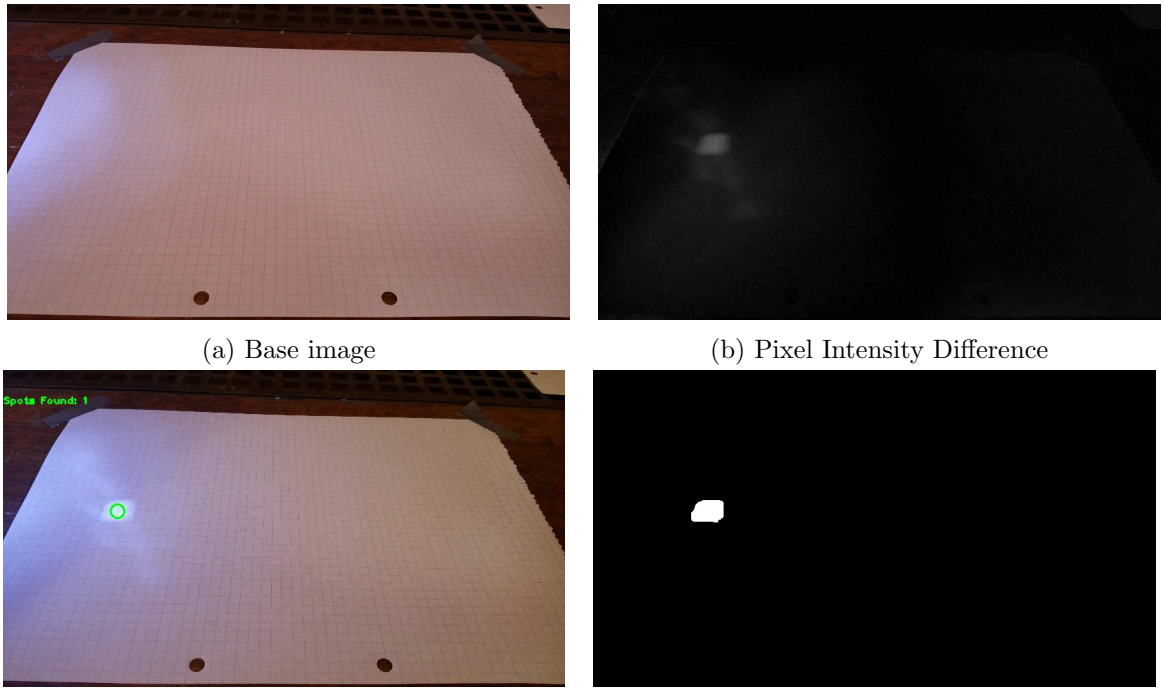
### 4.2.2 Test 2

This test was done with ambient light and with a single spot reflected off of a mirror.




(a) HSV = [100-120, 50-70, 240-255] (b) After Threshold.

Figure 12: Software results of the colour tracking algorithm for test 2.



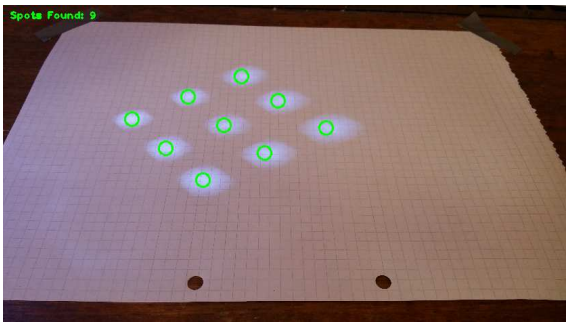

(a) Base image (b) Pixel Intensity Difference

(c) Sensitivity = 40. (d) After Threshold.

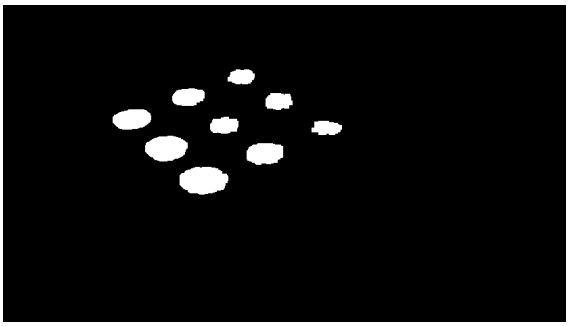
Figure 13: Software results of the difference tracking algorithm for test 2.

### 4.2.3 Test 3

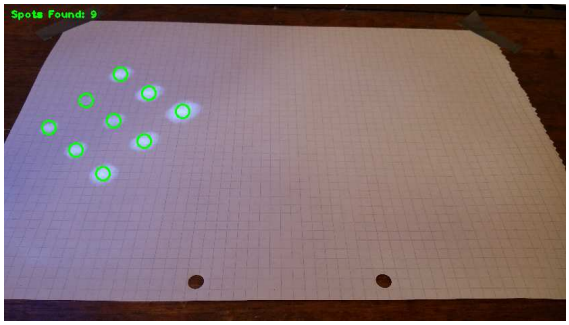
This test was done with ambient light and multiple spots reflected off of a mirror.



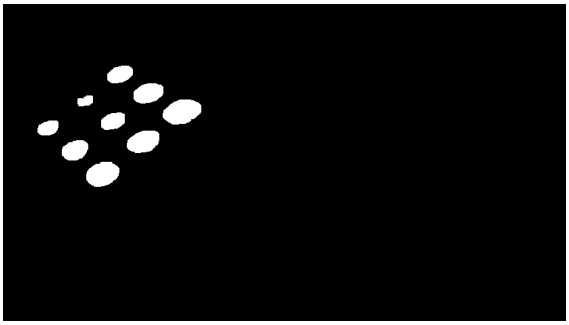

(a) HSV = [80-130, 45-85, 190-255]



(b) After Threshold.

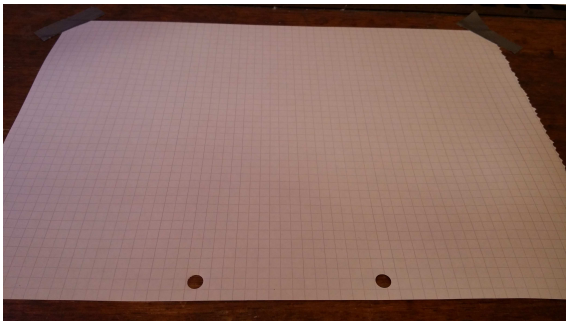



(c) HSV = [80-130, 45-85, 190-255]

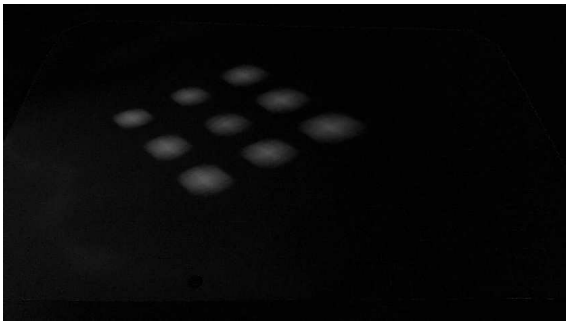


(d) After Threshold.

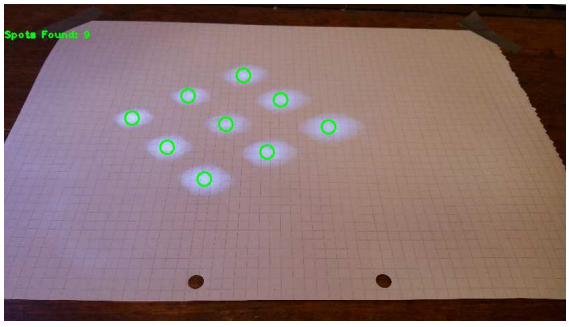
Figure 14: Software results of the colour tracking algorithm for test 3.



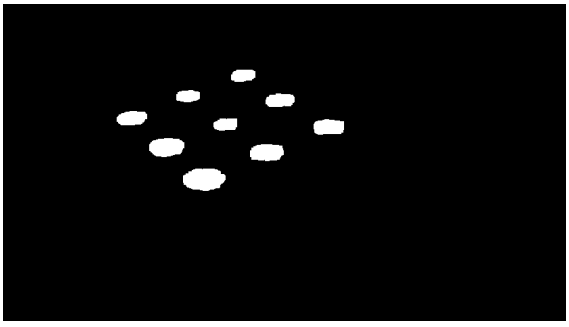
(a) Base image



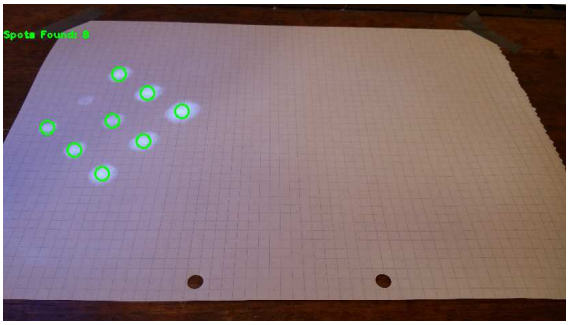
(b) Pixel Intensity Difference




(c) Sensitivity = 40.

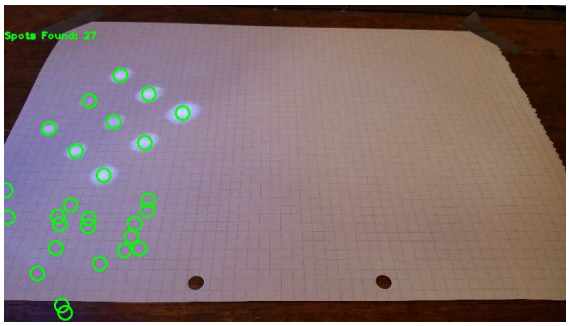


(d) After Threshold.




(e) Sensitivity = 40.

(f) After Threshold.



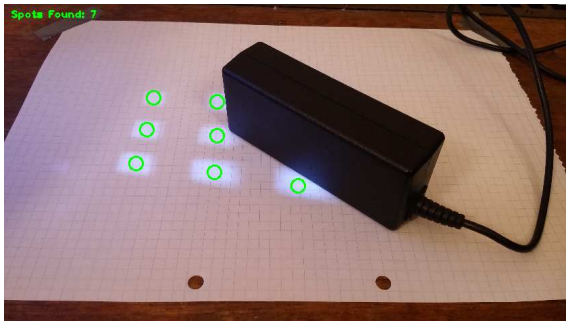

(g) Sensitivity = 20.

(h) After Threshold.

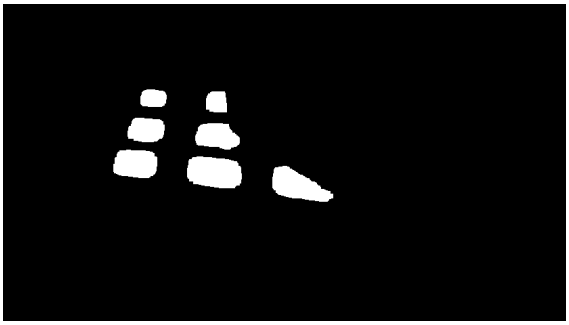
Figure 15: Software results of the difference tracking algorithm for test 3. Shown here is the effect of reducing the sensitivity to try and find dim spots.

#### 4.2.4 Test 4

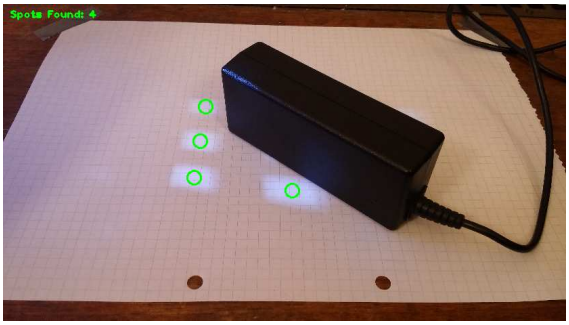
This test was done with ambient light and multiple spots reflected off of a mirror, with the addition of a black plastic object (a laptop charger) to the scene.




(a) HSV = [85-150, 30-70, 200-255]

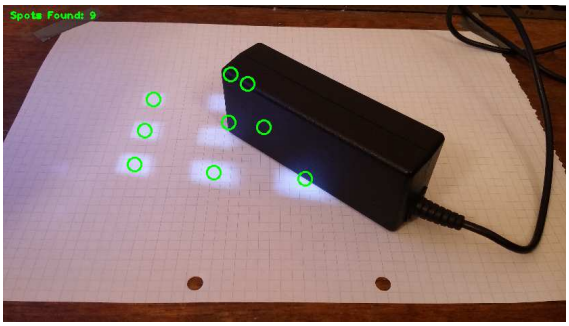


(b) After Threshold.



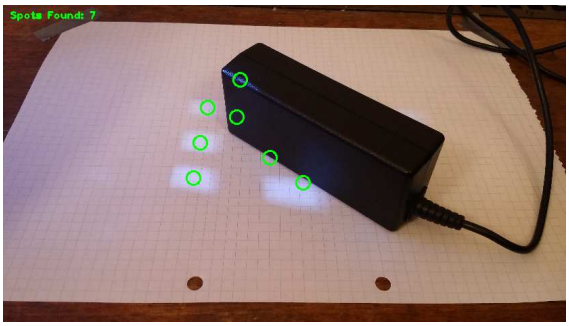

(c) HSV = [85-150, 30-70, 200-255]

(d) After Threshold.



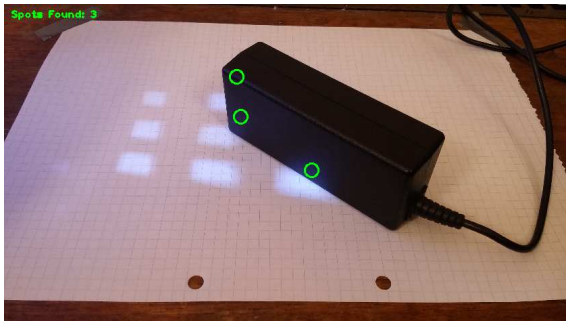

(e) HSV = [80-130, 30-255, 45-255]

(f) After Threshold.



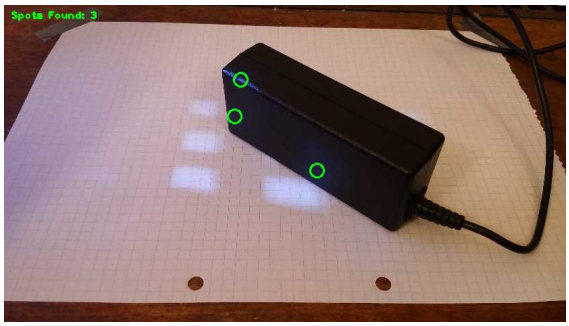

(g) HSV = [80-130, 30-255, 45-255]

(h) After Threshold.




(i) HSV = [80-130, 60-255, 45-255]

(j) After Threshold.

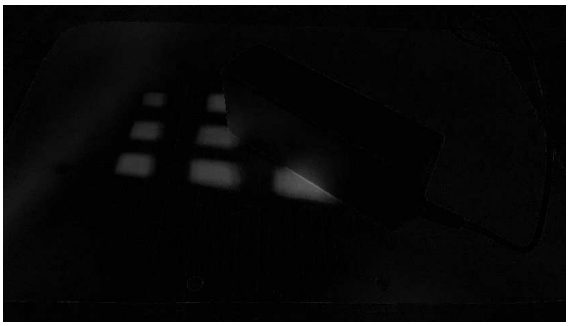



(k) HSV = [80-130, 60-255, 45-255]

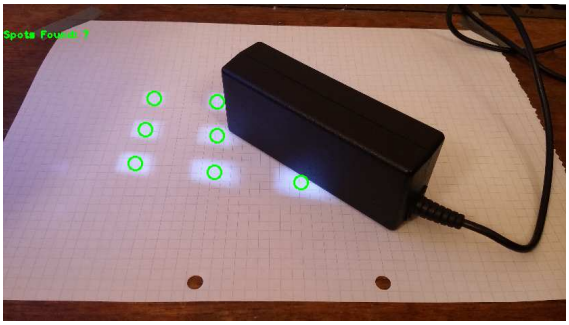
(l) After Threshold.

Figure 16: Software results of the colour tracking algorithm for test 4. Shown here is the attempt at finding light spots incident on a low-reflectivity object by varying the HSV threshold ranges.

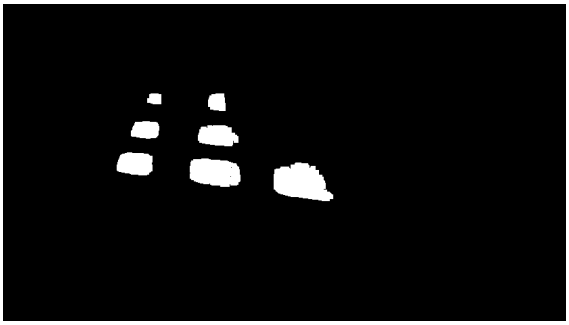
(a) Base image



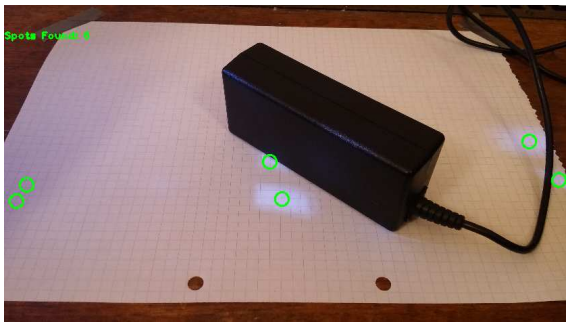
(b) Pixel Intensity Difference



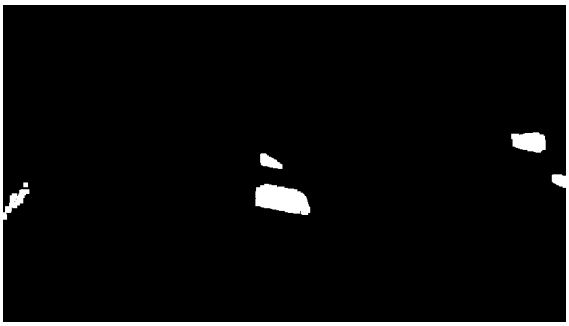
(c) Sensitivity = 20.



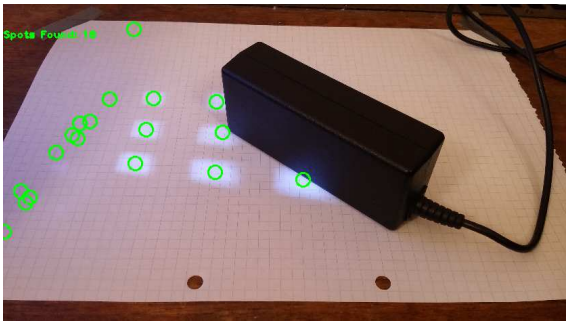
(d) After Threshold.



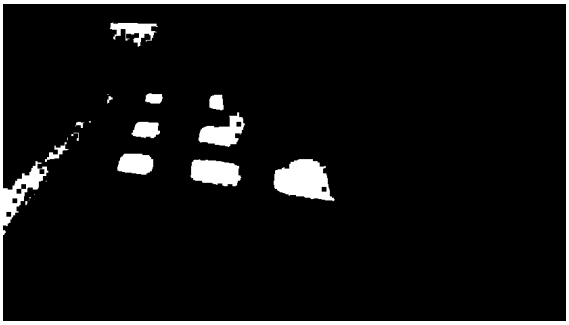
(e) Sensitivity = 20.



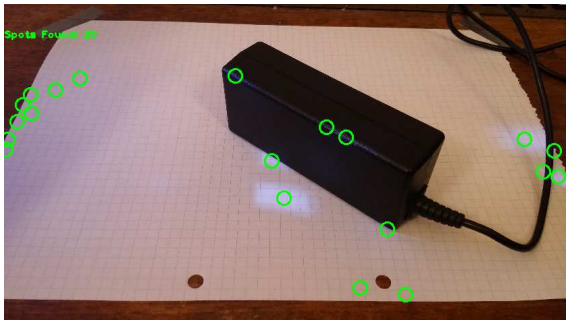
(f) After Threshold.



(g) Sensitivity = 10.



(h) After Threshold.



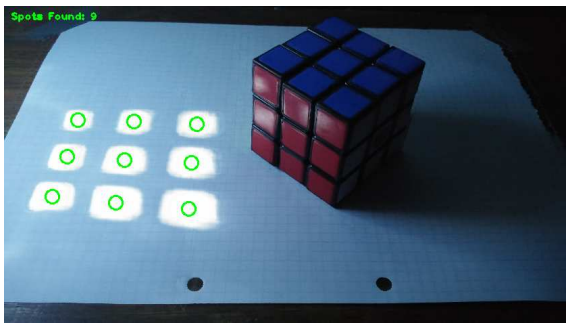
(i) Sensitivity = 10.

(j) After Threshold.

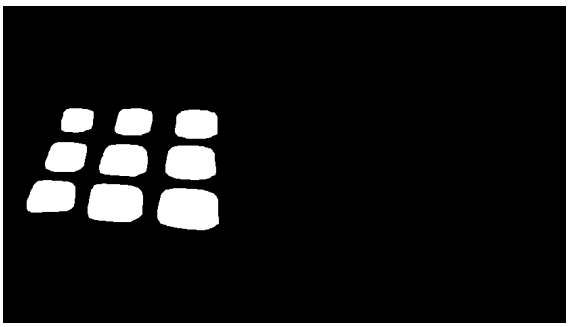
Figure 17: Software results of the difference tracking algorithm for test 4. Shown here is the attempt at finding light spots incident on a low-reflectivity object by reducing the sensitivity of the required intensity difference.

### 4.2.5 Test 5

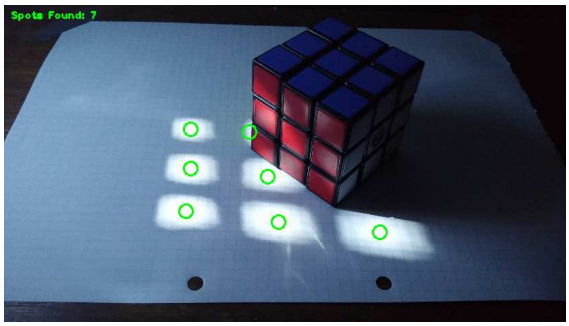
This test was done without ambient light, multiple spots reflected off of a mirror, and the addition of a multi-coloured cube.



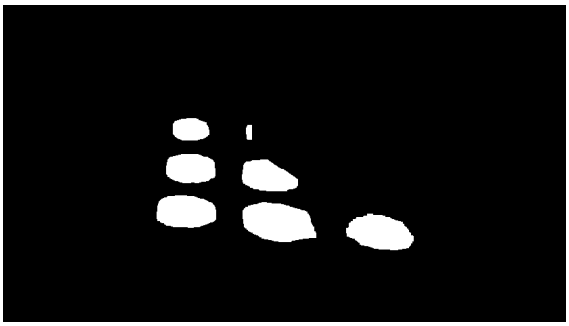

(a) HSV = [0-255, 0-40, 210-255]



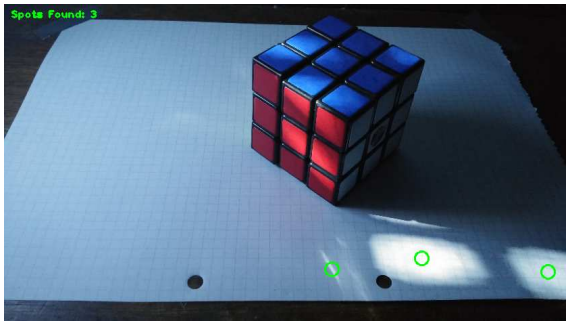
(b) After Threshold.




(c) HSV = [0-255, 0-40, 210-255]

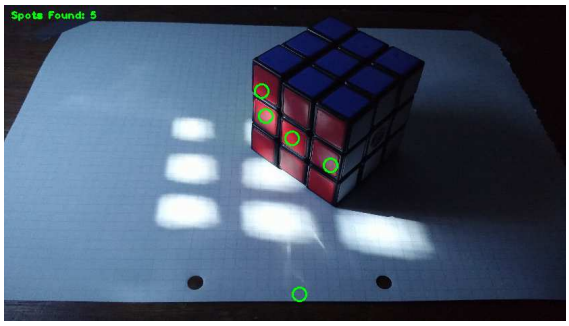


(d) After Threshold.

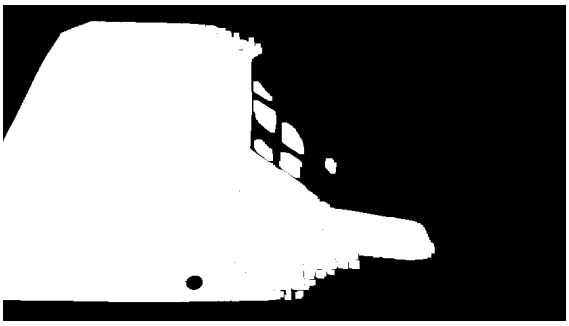



(e) HSV = [0-255, 0-40, 210-255]

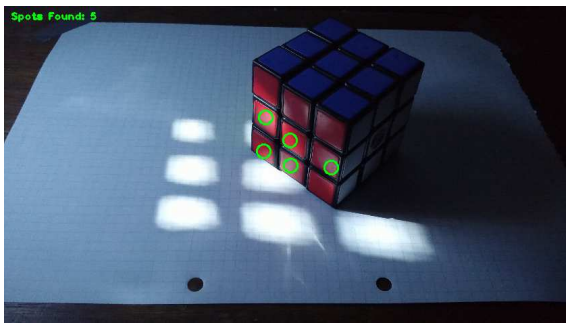
(f) After Threshold.



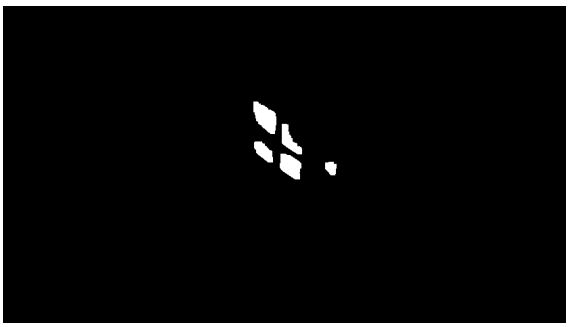

(g) HSV = [0-179, 0-255, 130-255]



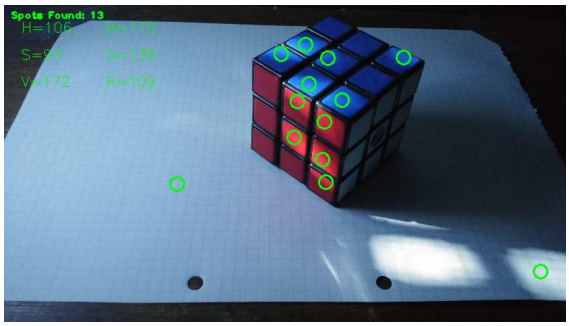
(h) After Threshold.



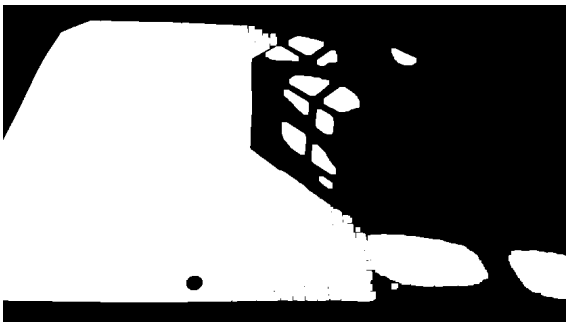

(i) HSV = [120-180, 30-210, 130-255]



(j) After Threshold.

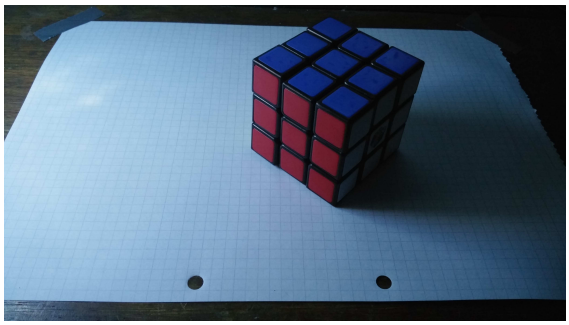



(k) HSV = [0-179, 0-255, 130-255]

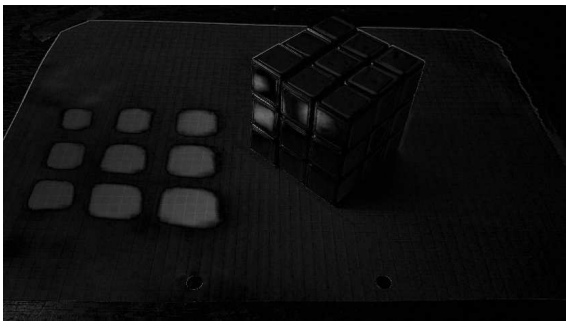


(l) After Threshold.

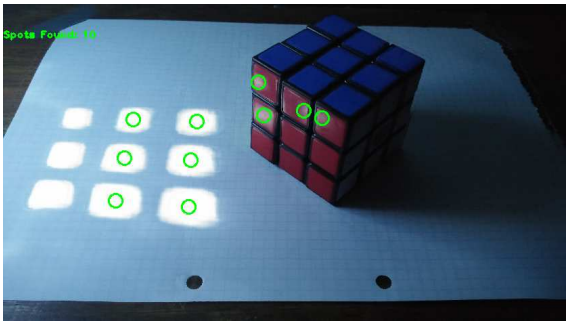
Figure 18: Software results of the colour tracking algorithm for test 5. Shown here is the attempt at finding light spots incident on a shiny multi-coloured cube by varying the HSV threshold ranges.



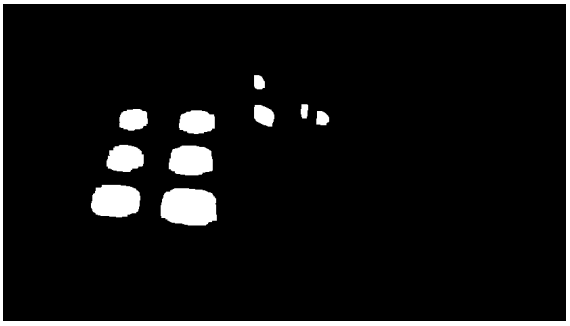
(a) Base image



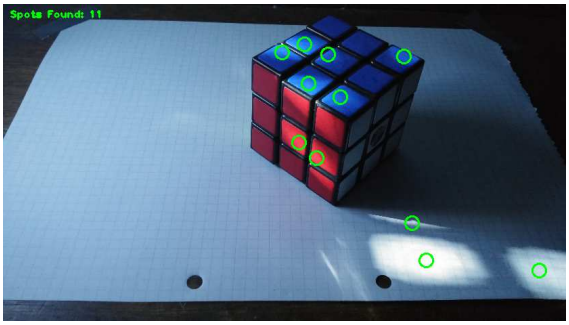
(b) Pixel Intensity Difference



(c) Sensitivity = 40.

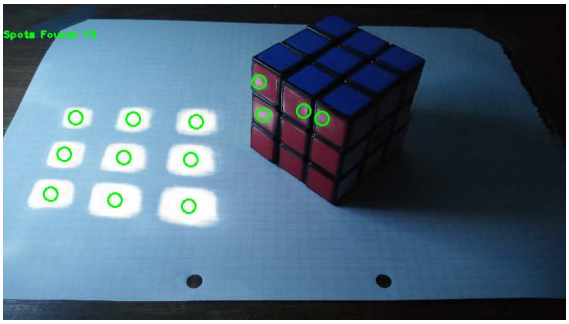


(d) After Threshold.

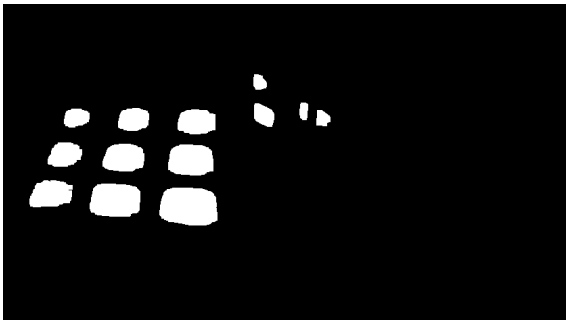



(e) Sensitivity = 40.

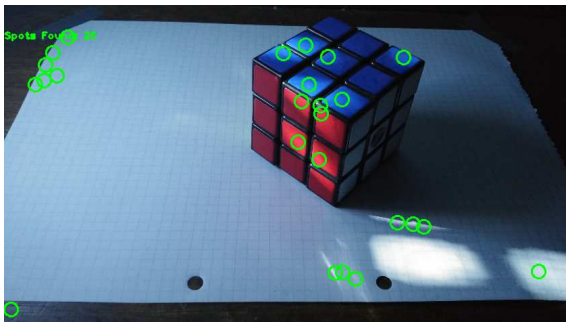
(f) After Threshold.



(g) Sensitivity = 30.



(h) After Threshold.



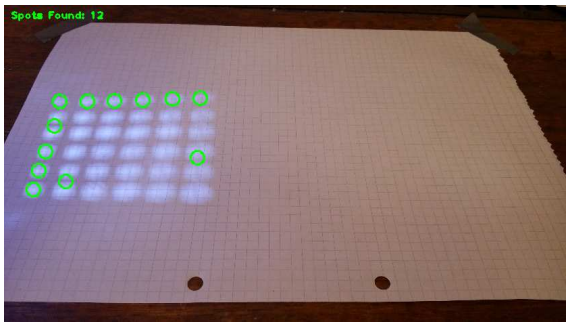
(i) Sensitivity = 30.

(j) After Threshold.

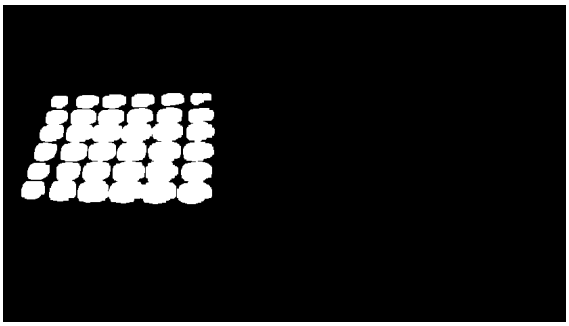
Figure 19: Software results of the difference tracking algorithm for test 5. Shown here is the attempt at finding light spots incident on a shiny multi-coloured cube by reducing the sensitivity of the required intensity difference.

### 4.2.6 Test 6

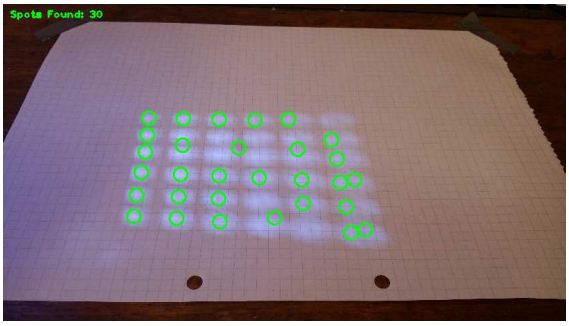
This test was done with ambient light and lots of spots being reflected off of a mirror.



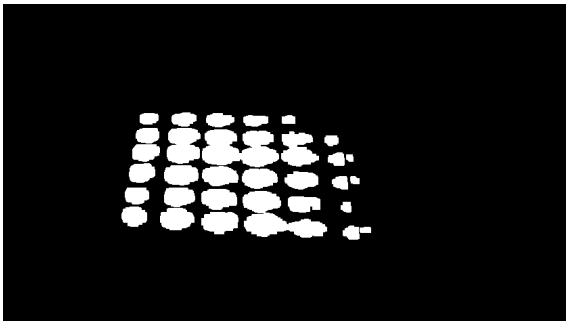

(a) HSV = [0-179, 0-100, 190-255]



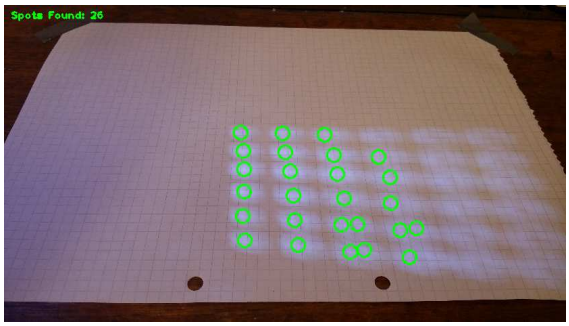
(b) After Threshold.



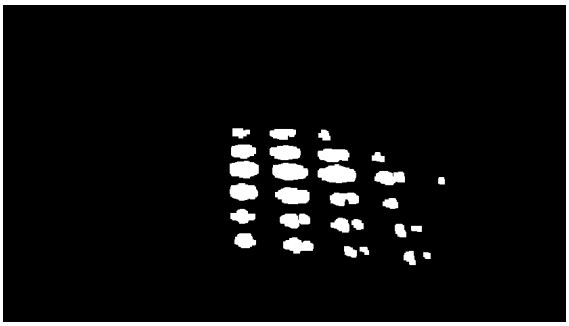

(c) HSV = [0-179, 0-100, 190-255]



(d) After Threshold.

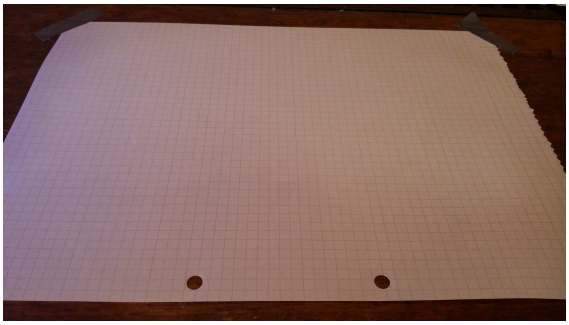



(e) HSV = [0-179, 0-100, 190-255]

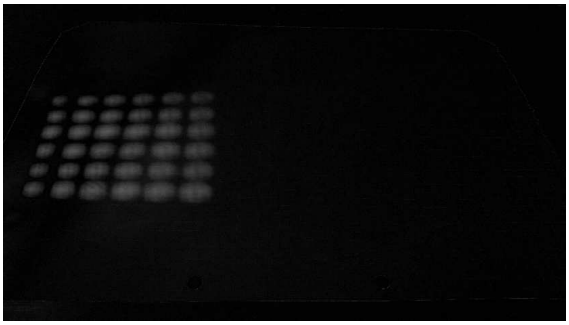


(f) After Threshold.

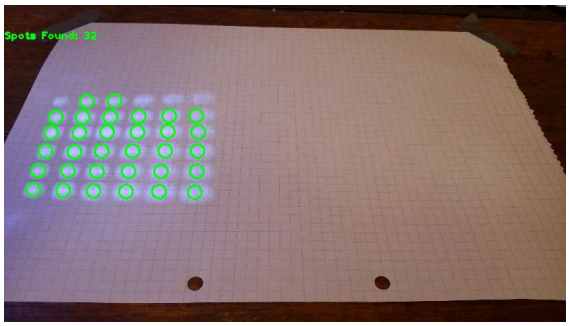
Figure 20: Software results of the colour tracking algorithm for test 6. Shown here is the attempt at finding multiple light spots that are small and close together.



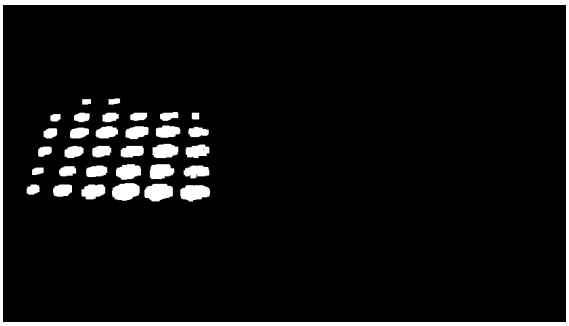
(a) Base image



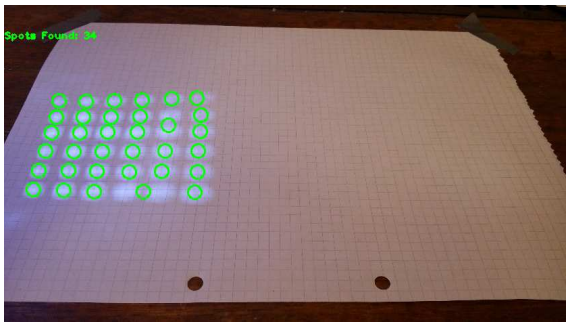
(b) Pixel Intensity Difference



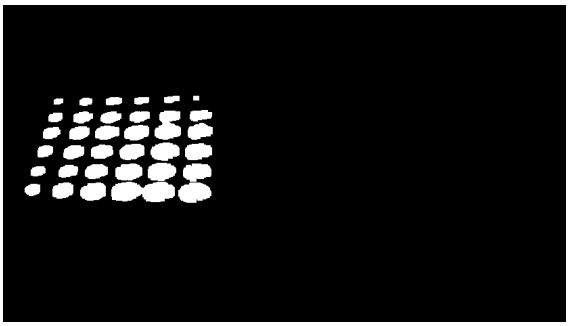

(c) Sensitivity = 40.



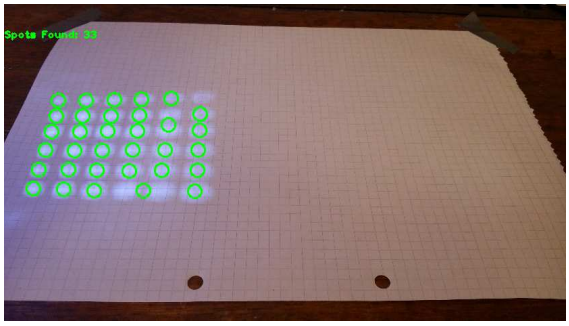
(d) After Threshold.



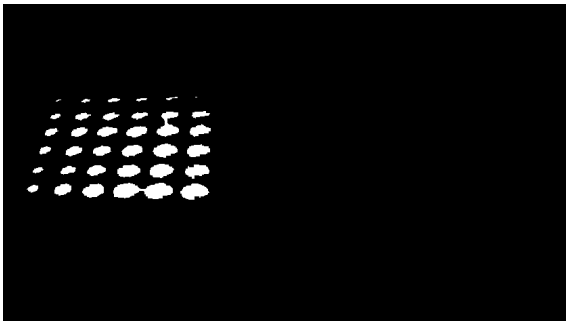

(e) Sensitivity = 20.



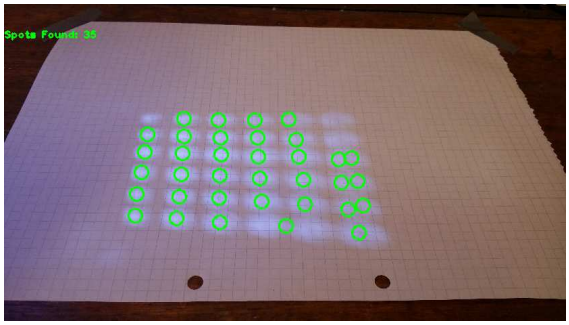
(f) After Threshold.



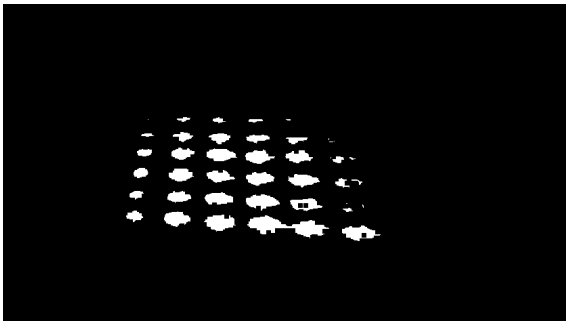

(g) Sensitivity = 20, no dilation.



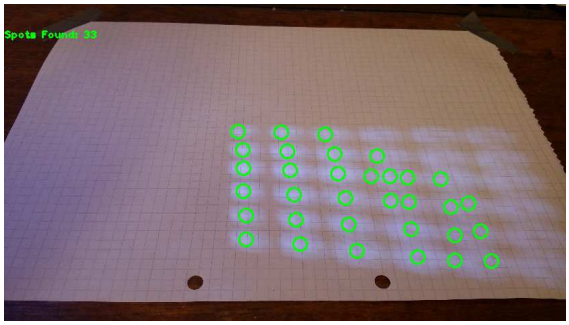
(h) After Threshold.



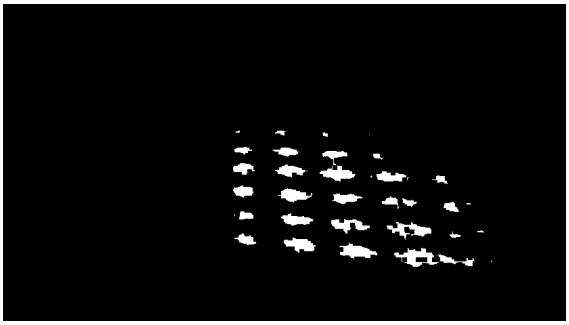

(i) Sensitivity = 20, no dilation.



(j) After Threshold.




(k) Sensitivity = 20, no dilation.

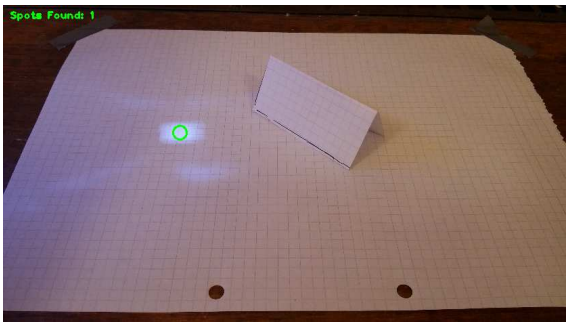


(l) After Threshold.

Figure 21: Software results of the difference tracking algorithm for test 6. Shown here is the attempt at finding multiple light spots that are small and close together.

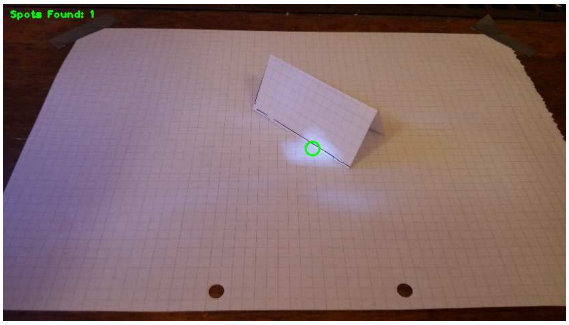
### 4.2.7 Test 7

This test was done with ambient light with a single spot being reflected off of a mirror and a simple paper prism in the scene.



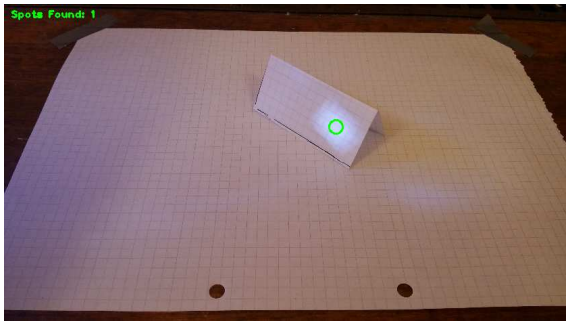

(a) HSV = [0-179, 0-70, 210-255]

(b) After Threshold.




(c) HSV = [0-179, 0-70, 210-255]

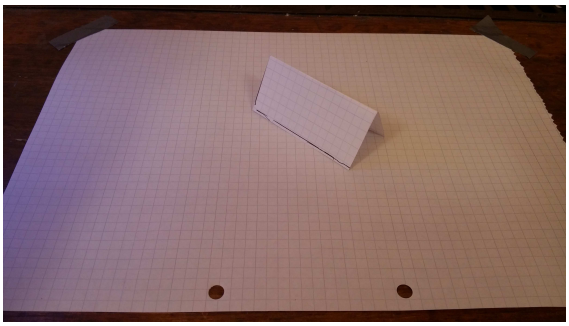
(d) After Threshold.




(e) HSV = [0-179, 0-70, 210-255]

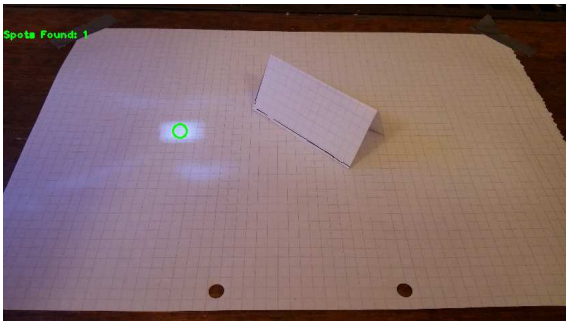
(f) After Threshold.

Figure 22: Software results of the colour tracking algorithm for test 7. Shown here is the attempt at finding a single light spot moving over a simple object.

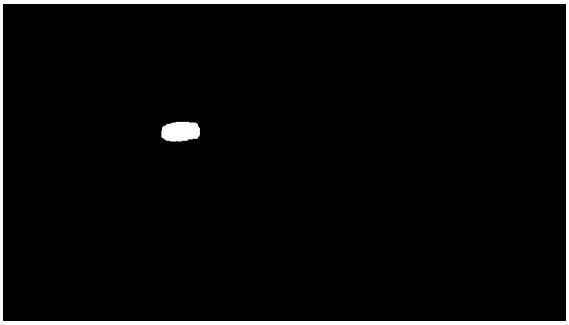


(a) Base image

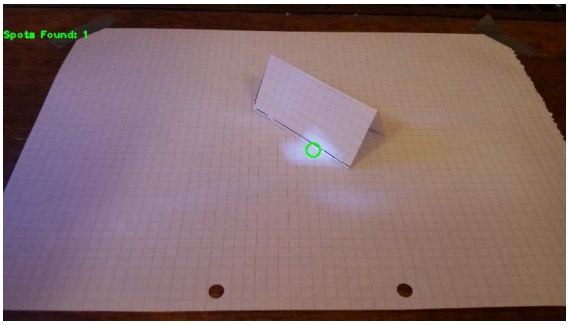
(b) Pixel Intensity Difference




(c) Sensitivity = 40.

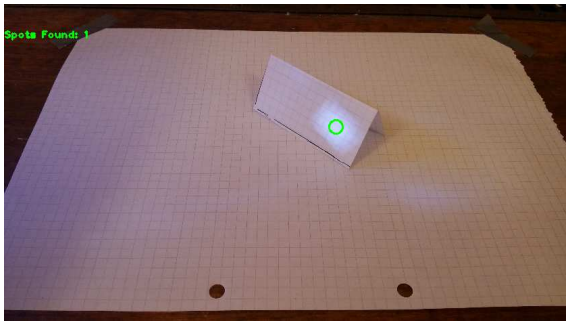


(d) After Threshold.




(e) Sensitivity = 40.

(f) After Threshold.




(g) Sensitivity = 40.

(h) After Threshold.

Figure 23: Software results of the difference tracking algorithm for test 7. Shown here is the attempt at finding a single light spot moving over a simple object.

# 5 Analysis & Discussion

Although both experiments were not perfectly successful, some very useful information was obtained that would help in the further development of a PSL system.

## 5.1 Geometry Results

The positions of the spot, mirror and light source were used to calculate vectors from the mirror to light source and from mirror to spot (Table 1). The mirror to spot (surface and cube) vectors (Table 3) are known because the spot positions could be measured, but would not be so in a real system. Instead, the vector from the light source to the mirror was reflected about the mirror normal to get the vector that should represent the reflected light from mirror to spot (Table 2).

After normalizing these vectors (known and calculated) it can be seen that they are roughly the same in both cases, whether there was an object in the way of the reflected light or not. Therefore by reflecting light from a mirror, we can calculate a vector that intersects surfaces in the environment and use them to discover where they intersect. This final step of triangulation to discover where the vector intersects the environment was not performed here.

## 5.2 Software Results

The output of the software of a real Passive Structured Light system would be the coordinates of the spots. However the ability of the software to actually find the spots is important, and as such the ability of the algorithms to do so is discussed here.

### 5.2.1 Test 1 - Single spot with no ambient light

In this ideal situation (Figures 10 & 11), where the spot of light is white and very bright, the spot should have a HSV range of [0-255, ≈0, ≈255]. This setting was tested (Figure 10a) and had no problems locating it, and since the bright white spot created a large increase in pixel intensity compared to the base image (with no spot), the difference method had no trouble locating the spot either (Figure 11c).

### 5.2.2 Test 2 - Single spot with ambient light

For this test, there was the addition of ambient light (Figures 12 & 13). This had two main effects: the light was tinted slightly, and the spot was dimmer. Since the light was tinted, the hue value had to be more specific to not pick up unwanted light, and the dimmer spot meant that the saturation value had to be higher and have a wider range (Figure 12a). The wider ranges of values meant that more background noise was picked up, but with careful tweaking of the parameters the spots could be isolated. One problem that was found during this test was that the spot became dimmer and more distorted as the distance from the mirror increased, which meant that as the spot got further away, the parameters required to find it changed.

The difference method had no trouble once again finding the spot, as it was bright enough to make a noticeable difference to the base image (Figure 13c). As the spot became more distorted

and dimmer with distance, this difference became smaller, especially around the edges, but the spot could still be found without finding any background noise.

#### 5.2.3 Test 3 - Multiple spots with ambient light

Test 3 involved multiple spots in the presence of ambient light, but the results were similar to that of when there was just a single spot (Figures 14 & 15). The spots could be found with wider HSV ranges, but not so wide that background noise was picked up. Since these spots were spread out, the distance of each from the mirror was different, and thus the amount of distortion & dimming between each spot varied. This caused the HSV ranges to be wider than before, but it still did not pick up any background noise.

Similar to the previous test, the difference method could still find all the spots despite the decrease in brightness and varied distortion between spots. However, there was one anomalous result in this test, which is that one of the top-left spots in the second pair of images (Figure 14c & 15e) was much dimmer than the rest. This did not cause a problem for the HSV method, since it already allowed for a wider range of brightnesses. However the difference method could not find this dim spot (by lowering the sensitivity) without picking up background noise (Figure 15g & 15h).

#### 5.2.4 Test 4 - Multiple spots with ambient light and a black plastic object

This test involved multiple spots once again, however a black plastic object (a laptop charger) was introduced and the spots of light were moved across it to see if they could still be detected (Figures 16 & 17). With the same parameters as before, the spots on the paper could be found, but not on the object (Figure 16a). In an attempt to find all the spots (on both paper and object) at the same time, the parameters were changed, but it could not be done accurately.

The spots ended up making very small differences to the brightness of the black object, and by increasing the range of the parameters, too much background noise was picked up. The small differences that could be picked up were quite wide and inaccurate, which caused multiple spots to be seen as a single one (Figure 16e & 16g). Specular reflections off of the corner of the object were also detected and identified as a spot, which is unwanted.

In the event of this occurring in a real system, it might be possible to expect these different surfaces and to account for them by isolating the spots on the object & surface separately and then merging the results together. An attempt was made at trying to find the spots on only the object, which performed well (Figure 16i & 16k), but as before it picked up unwanted specular reflections. Since the object was at an angle, the spots also became more distorted, thus making them less accurate and harder to find.

The significantly dimmer spots on the object caused problems for the difference finding method. The spots on the paper could be picked up consistently, like before, but the spots on the object only caused tiny changes to the intensity of the pixels, and so could not be found with the previous settings (Figure 17c). In an attempt to locate them, the sensitivity was lowered (Figure 17g & 17i), but this picked up lots of other minute differences in the background as noise. In addition to this, the algorithm picked up the specular reflections off of the corner of the object, which is unwanted.

### 5.2.5 Test 5 - Multiple spots with no ambient light and a multicoloured object

Test 5 replaced the black object with a multi-coloured cube (a rubiks cube) and removed the ambient light (Figures 18 & 19). The spots were very bright and could easily be found like before, but there was still difficulty finding the spots when they were incident on the cube (Figures 18a & 18c & 18e). Since the spots were bright, the hue parameters did not matter, however when they were incident on the cube they were tinted red / blue and became much more saturated and dimmer; therefore the saturation and value parameters were higher and lower respectively. In an attempt to find all the spots at the same time, the saturation and value ranges were increased, but this caused a large amount of background noise to be included and the spots on the paper were untraceable (Figures 18g & 18k).

As before, an attempt was made to isolate the spots on the cube face. Since the cube faces were distinct colours, this was quite easy as the hue parameters could be changed to isolate specific colours. However the black lines between the squares of the cube caused single spots cast over multiple squares to be picked up as multiple spots (Figure 18i). Since the cube faces were different colours, spots present on the various sides could not be found at the same time.

The difference method could accurately find spots on the paper due to the brightness compared to the background. While in the image the spots appear bright, the difference between them and the base image (Figure 19a) is not that large, as can be seen in the difference image (Figure 19b), which caused some of the spots to initially not be found (Figure 19c).

This was due to there being too much general illumination in the base image. This illustrates two problems with this method: the base image is important and needs to accurately represent the environment without intrusion from the apparatus, and light spots might not increase the intensity of bright objects enough to think there is a spot on them. An attempt to account for this was done by lowering the sensitivity, but due to this mistake background noise was picked up later in the tests (Figures 19g & 19i).

When the spots were cast over the cube, the difference method could find them at the same time as the spots on the paper, and could find them on both of the cube faces at the same time. However, the same problem as before occurred, whereby the black lines separating the squares caused single spots to register as multiple (Figures 19e & 19i). In addition to this, unwanted specular reflections in the cube face were picked up (Figures 19g & 19c).

While in a real application of this techniques, it would not be very likely to find objects with colours like that of a rubiks cube (bright blue & red), this test helps to show that light spots change based on the surface they are reflected upon. It also shows that objects that are not simply one colour and have more complexities to them (ie the black lines between the squares) need to be accounted for.

### 5.2.6 Test 6 - Many spots with ambient light

This test reflected many more light spots onto the environment with ambient light (Figures 20 & 21). In a real application of this technique, it would be ideal for many spots to be used in order to gather data quickly, and for the spots to be small to make the measurements more precise. This was investigated in this test.

During the test, the issue of distortion became a problem because of the close proximity of all the spots. While the spots could be isolated in the HSV-thresholding step, the fact that they

were dim and blurred at the edges meant that the parameter ranges had to be large to find them all. This meant that the resulting white areas were large and tended to overlap with each other, causing multiple spots to be seen as one (Figure 20b). In an attempt to fix this, the amount of dilation was reduced. However this caused the more dimmer spots to disappear completely (Figure 20d), since some were more distorted / blurred than others (due to the distance from the mirror). Eventually, the blurring & distortion due to the angle & distance of reflection caused the spots to be indistinguishable to the algorithm without introducing a lot of background noise (Figure 20f).

The difference algorithm did not perform any better than the HSV method and was very inaccurate. Due to distortion and blurring, some spots were not picked up initially, however the spots that were found did not overlap as much as before (Figure 21c). In an attempt to find the missing spots, the sensitivity was decreased, which succeeded in finding them, but caused other spots to enlarge and merge together (Figure 21e). Dilation was then turned off, like before, to try and stop this merging. This final setting was barely able to find all the spots (Figure 21g), but could not do so reliably as the spots were moved across the paper (Figures 21i & 21k) and the distortion became more significant. Dilation was required to make these dimmer spots more pronounced.

This test has shown that mirrors used to reflect the light need to be designed well to make the process of finding the spots easier. While it is important for there to be lots of spots, they either need to be further apart to be distinguishable from one another, or much smaller and brighter.

#### 5.2.7 Test 7 - Single spot with ambient light and small white object

This final, simple test was done to test the algorithms moving over a simple object with the same properties as the paper surface (Figures 22 & 23). Both the hsv (Figures 22a - 22e) and difference (Figures 23c - 23g) methods had no trouble finding the single spot, without any background noise, as it moved over the prism.

### 5.3 General Results Discussion

In many cases during the tests, the parameters of the program had to be tweaked manually to correctly find the spots. For a fully developed PSL system, it could be done manually. However for an extra-terrestrial robot it would most likely need to be done automatically. This could be done by pre-programming in the expected qualities of the spots, predicting any colour-tinting that may occur to the spots as they move across different surfaces, or predicting the effect of weather conditions on the brightness of the spots.

The spots were frequently affected by distortion due to the distance from the mirror. This could cause problems in the techniques application, as the distortion could become great enough at some distance whereby the spot of no longer discernible - therefore giving this technique a limited range. However, since the distortion was proportional to the distance from the mirror, this relationship could be used to determine some minor range information very quickly.

In order to create a precise model using the spots, a vector from each mirror to the corresponding spot is required. However in this experiment it would have been difficult to match each spot to the mirror it came from. A method of discerning which spot came from which mirror would be needed. This could be done by changing the shape of the mirror to get spots of varying shapes, or by tinting the mirrors to get different coloured spots.

# 6 Conclusion

While this experiment did not result in a fully functional Passive Structured Light system, some useful information can be gained from it. The two methods of tracking spots in the image were tested and showed promise in simple situations, but need improvement if they are to become more robust and reliable. Such improvements could be to merge the two techniques together to help find spots even with the presence of background noise, or to add other criteria for identifying spots, such as looking for a specific shape.

The tests done in this experiment have not been specifically focused at testing parts of the technique in a realistic planetary environment (e.g. Mars). However, some of the issues raised regarding the application of the spot tracking techniques are still applicable. For example, while images of Martian environments show us a generally bland and monochromatic world, much of the environment will be of different colours / shades and have varying material properties, like the objects used in the tests. This could factor into the effectiveness of the technique on another planet.

In addition to this, the environment could have an effect on this technique as well, more so than it would on other techniques. This passive version of structured light requires use of the sun. On other planets, the sun will be a different distance away (most likely further) and so the incident light will be dimmer and the spots will be harder to track. The effects of other environmental factors, such as clouds blocking the sun, or dust settling on the mirrors would need to be investigated as well.

This experiment has provided a starting point in the investigation of the quality of this technique. More common methods, such as stereo vision, continue to be used for planetary robots because of their reliability and robustness, which is very important in the unknown extra-terrestrial environments. However if some of the problems of PSL outlined here can be overcome, then it could serve as an alternative option due to its advantages in areas of power use, weight, required electronics, and other spin-off applications.

# References


[1] Various. Open Source Computer Vision Library. itseez, 2015. Accessed: February 2015, Available at `http://opencv.org/`.

[2] Hounslow, Kyle. OpenCV and Object Tracking Tutorials, June 2014. Accessed: February 2015, Available at `https://www.youtube.com/playlist?list=PLvwB65U8V0HHCEyW2UTyOJym5FsdqfbHQ`.

[3] Joaquim Salvi, Jordi Pags, and Joan Batlle. Pattern codification strategies in structured light systems. *PATTERN RECOGNITION*, 37:827 – 849, 2004.

[4] Jason Geng. Structured-light 3d surface imaging: a tutorial. *Advances in Optics and Photonics*, 3(2):128–160, Jun 2011.

[5] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195. IEEE, 2003.

[6] Chu-Song Chen, Yi-Ping Hung, Chiann-Chu Chiang, and Ja-Ling Wu. Range data acquisition using color structured lighting and stereo vision. *Image and Vision Computing*, 15(6):445–456, 1997.

[7] Clarke Olson and Larry Matthies and John Wright and Ron Li and Kaichang Di. Visual Terrain Mapping for Mars Exploration. *Computer Vision and Image Understanding*, (105):73 – 85.

[8] Rajesh Rao and Jiun-Hung Chen. Stereo and 3D Vision. University of Washington Computer Science and Engineering: `https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf`, 2009. Accessed: Novermber 2014.

[9] Yasuharu Kunii and Takahiro Ushioda. Shadow casting stereo imaging for high accurate and robust stereo processing of natural environment. In *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pages 302–307, July 2008.

[10] Anthony Cook. Do we really need to put Stereo Cameras on Landers? 2007.

[11] Heiko Hirschmuller and Peter Innocent and Jon Garibaldi. Fast, Unconstrained Camera Motion Estimation from Stereo without Tracking and Robust Statistics. In *International Conference on Control, Automation, Robotics and Vision (ICARCV'02)*, volume VII, pages 1099–1104, Singapore, December 2003.

[12] David Murray and Paul Beardsley. Range Recovery using Virtual Multi-Camera Stereo. In *In Proceedings of the British Machine Vision Conference*, pages 29 – 38, Leeds, September 1992.

[13] Hartmut Surmann and Andreas Nuchter and Joachim Hertzberg. An Autonomous Mobile Robot with a 3D Laser Range Finder for 3D Exploration and Digitalization of Indoor Environments. *Robotics and Autonomous Systems*, (45):181–198.

[14] David Cole and Paul Newman. Using Laser Range Data for 3D SLAM in Outdoor Environments. In *In Proceedings of the 2006 International Conference on Robotics and Automation*, pages 1556 – 1563, Orlando, Florida, May 2006.

[15] Liang Lu, Camilo Ordonez, Emmanuel G. Collins Jr, and Edmond M. DuPont. Terrain surface classification for autonomous ground vehicles using a 2d laser stripe-based structured light sensor. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 2174 – 2181. IEEE, 2009.

[16] Filip Sadlo and Tim Weyrich and Ronald Peikert and Markus Gross. A Practical Structured Light Acquisition System for Point-Based Geometry and Texture. In M. Pauly and M. Zwicker, editor, *Proceedings of Eurographics Symposium on Point-Based Graphics*, 2005.

[17] Miguel Ribo and Markus Brandner. State of the art on vision-based structured light systems for 3d measurements. In *Robotic Sensors: Robotic and Sensor Environments, 2005. International Workshop on*, pages 2–6, Sept 2005.

[18] Yasuharu Kunii and Taeko Gotoh. Evaluation of shadow range finder: Srf for planetary surface exploration. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 2573 – 2578, Sept 2003.

[19] NASA. Phoenix Mission Overview. NASA: `http://www.nasa.gov/mission_pages/phoenix/spacecraft/index.html#.VGyo54U3jFb`, 2013. Accessed: November 2014.

[20] Patrick Blau. Chang'e-3. Spaceflight101: `http://www.spaceflight101.com/change-3.html`, 2014. Accessed: November 2014.

[21] Jet Propulsion Laboratory. Mars Pathfinder Mission. NASA JPL: `http://mars.jpl.nasa.gov/MPF/mpf/sci_desc.html#IMP`, 1997. Accessed: December 2014.

[22] Michael Malin. MSL Mast Camera (MastCam). Malin Space Science Systems: `http://www.msss.com/all_projects/msl-mastcam.php`, 2011. Accessed: December 2014.

[23] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, and Andres Huertas. Computer vision on mars. *International Journal of Computer Vision*, 75(1):67–92, 2007.

[24] Edward Tunstel and Ayanna Howard. Sensing and perception challenges of planetary surface robotics, 2002.

[25] Narunas Vaskevicius, Andreas Birk, Kaustubh Pathak, and Sören Schwertfeger. Efficient representation in 3d environment modeling for planetary robotic exploration. *Advanced Robotics*, 24(8-9).

[26] David Ilstrup and Gabriel Elkaim. Low Cost, Low Power Structured Light Based Obstacle Detection. In *Position, Location and Navigation Symposium*, pages 771 – 778, Monterey, CA, May 2008.

[27] Anthony Cook and David Barnes. Terrain Mapping Coverage around a Lander using Passive Structured Light. In *Lunar and Planetary Science*, volume XXXIX, 2008.

# A Appendix

## A.1 Spot Tracking Code

The sections below display the source files and code for the light spot tracking program used in this report. The program was split into three source files: 'main.cpp', 'MorphOps.h', and 'MorphOps.cpp'.

### A.1.1 main.cpp

The source file 'main.cpp' contains the entry point (main method) of the program, controls the flow and routine of the program, and has the implementations of the two spot tracking algorithms within. The functions that contain the spot tracking code are called 'trackByDifference', 'trackByColour', and 'trackThresholdPixels'.

```
/*
 * main.cpp
 *
 *  The main source file containing functions to control the flow of the program and the main functionality of the program - the
 *  tracking of light spots.
 *
 *  Created on: 5 Feb 2015
 *      Author: Spencer Newton
 *
 *  Algorithms and Code based on tutorials by Kyle Hounslow
 */

// Includes
#include "opencv2/video/tracking.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "Globals.h"
#include "MorphOps.h"
#include "Geometry.h"
#include <stdio.h>
#include <vector>
using std::vector;

using namespace cv;
using namespace std;

// ============================== Variables ============================== //
// Image Names / List
const int IMAGE_WIDTH = 640;
const int IMAGE_HEIGHT = 480;
const int imageIndexMax = 6;
const int imageSetIndexMax = 7;
char* imageNames[imageSetIndexMax][imageIndexMax];
int imageIndex = 0;
int imageSetIndex = 0;

// Window Names
char mainWindowName[] = "Original Image";
char hsvWindowName[] = "HSV Morph";
char thresholdWindowName[] = "Thresholded Feed";
char trackbarWindowName[] = "Trackbars";
```

```
41 char differenceWindowName[] = "Frame Difference";
42 char differenceThresholdWindowName[] = "Difference Threshold";
43
44 // Parameter to store video feed
45 VideoCapture videoCapture;
46 const int VIDEO_WIDTH = 640;
47 const int VIDEO_HEIGHT = 480;
48
49 // Matrices to store frames
50 Mat frame, nextFrame; // Raw frames from camera / image; nextFrame for motion
       tracking comparison
51 Mat frameGray, nextFrameGray; // Gray frames for motion tracking
52 Mat hsvFrame; // Matrix to store HSV colour conversion
53 Mat thresholdFrame; // Matrix to store thresholded HSV image
54 Mat differenceFrame; // Matrix to store pixel differences between two frames
55 Mat differenceThresholdFrame; // Matrix to store thresholded difference image
56
57 // Motion Thresholding Parameters
58 int thresholdSensitivity = 40;
59 int thresholdSensitivityMax = 255;
60
61 // HSV Thresholding Parameters
62 int hMin = 0, sMin = 0, vMin = 0;
63 int hMax = 179, sMax = 255, vMax = 255;
64
65 // Morphological Operation Parameters
66 int erodeSize = 3, erodeMax = 9;
67 int dilateSize = 3, dilateMax = 9;
68 int blurStrength = 0, blurMax = 20;
69
70 // Tracking Parameters
71 int numberOfObjects;
72 int maxNumberOfObjects = 50;
73 int objectAreaMin = 10 * 10;
74 int objectAreaMax = (100 * 100);
75
76 // Control Parameters
77 bool blurFrame = true, erodeFrame = true, dilateFrame = true, trackFrame = true;
78 bool videoTrack = false, imageTrack = true;
79 bool colourTrack = false, differenceTrack = false;
80 bool printCoordinates = false, showHSV = true;
81 int mouseX, mouseY;
82
83 int input = 0;
84
85
86 // ================================== End Variables ================================
       //
87
88
89
90 // ================================== Function Declarations
       ================================== //
91 void runTrackingAlgorithm();
92 void trackByColour();
93 void trackByDifference();
94 void setUpColourWindows();
95 void setUpMainWindow();
96 void setUpMotionWindows();
```

```
97  void setOdd( int, void *);
98  void trackThresholdPixels( Mat );
99  static void onMouse( int, int, int, int, void* );
100 void printHSV();
101
102 // ========================================= End Function Declarations
        ========================================= //
103
104 /*
105  * Converts and integer into a string
106  */
107 string intToString(int number)
108 {
109   std::stringstream ss;
110   ss << number;
111   return ss.str();
112 }
113
114 /*
115  * Initializes the vector of image filenames.
116  */
117 void setUpImageNames()
118 {
119   imageNames[0][0] = "Images/new/d_ss_0.jpg";
120   imageNames[0][1] = "Images/new/d_ss_1.jpg";
121   imageNames[0][2] = "Images/new/d_ss_1.jpg";
122   imageNames[0][3] = "Images/new/d_ss_1.jpg";
123   imageNames[0][4] = "Images/new/d_ss_1.jpg";
124   imageNames[0][5] = "Images/new/d_ss_1.jpg";
125
126   imageNames[1][0] = "Images/new/l_ss_0.jpg";
127   imageNames[1][1] = "Images/new/l_ss_1.jpg";
128   imageNames[1][2] = "Images/new/l_ss_2.jpg";
129   imageNames[1][3] = "Images/new/l_ss_3.jpg";
130   imageNames[1][4] = "Images/new/l_ss_4.jpg";
131   imageNames[1][5] = "Images/new/l_ss_5.jpg";
132
133   imageNames[2][0] = "Images/new/l_9s_0.jpg";
134   imageNames[2][1] = "Images/new/l_9s_1.jpg";
135   imageNames[2][2] = "Images/new/l_9s_2.jpg";
136   imageNames[2][3] = "Images/new/l_9s_3.jpg";
137   imageNames[2][4] = "Images/new/l_9s_4.jpg";
138   imageNames[2][5] = "Images/new/l_9s_5.jpg";
139
140   imageNames[3][0] = "Images/new/lc_9s_0.jpg";
141   imageNames[3][1] = "Images/new/lc_9s_1.jpg";
142   imageNames[3][2] = "Images/new/lc_9s_2.jpg";
143   imageNames[3][3] = "Images/new/lc_9s_3.jpg";
144   imageNames[3][4] = "Images/new/lc_9s_4.jpg";
145   imageNames[3][5] = "Images/new/lc_9s_4.jpg";
146
147   imageNames[4][0] = "Images/new/dcube_9s_0.jpg";
148   imageNames[4][1] = "Images/new/dcube_9s_1.jpg";
149   imageNames[4][2] = "Images/new/dcube_9s_2.jpg";
150   imageNames[4][3] = "Images/new/dcube_9s_3.jpg";
151   imageNames[4][4] = "Images/new/dcube_9s_4.jpg";
152   imageNames[4][5] = "Images/new/dcube_9s_5.jpg";
153
154   imageNames[5][0] = "Images/new/l_36s_0.jpg";
```

```
155   imageNames[5][1] = "Images/new/l_36s_1.jpg";
156   imageNames[5][2] = "Images/new/l_36s_2.jpg";
157   imageNames[5][3] = "Images/new/l_36s_3.jpg";
158   imageNames[5][4] = "Images/new/l_36s_4.jpg";
159   imageNames[5][5] = "Images/new/l_36s_4.jpg";
160
161   imageNames[6][0] = "Images/new/lp_ss_0.jpg";
162   imageNames[6][1] = "Images/new/lp_ss_1.jpg";
163   imageNames[6][2] = "Images/new/lp_ss_2.jpg";
164   imageNames[6][3] = "Images/new/lp_ss_3.jpg";
165   imageNames[6][4] = "Images/new/lp_ss_4.jpg";
166   imageNames[6][5] = "Images/new/lp_ss_5.jpg";
167
168 }
169
170 /*
171  * The main method of the program.
172  */
173 int main( int argc, char** argv)
174 {
175   char choice;
176   setUpImageNames();
177
178   cout << "Track or Calculate: t or c\n";
179   cin >> choice;
180
181   switch (choice)
182   {
183   case 't':
184     runTrackingAlgorithm();
185     break;
186
187   case 'c':
188     runGeometryCalculations();
189     break;
190
191   default:
192     break;
193   }
194 }
195
196 /*
197  * This function is the main loop of the program. It will continually load, modify,
       and display the images that are being used to try and find light spots.
198  *
199  *
200  * WAITKEY KEYCODES:
201  *   'a' = 97
202  *   'b' = 98
203  *   'c' = 99
204  *   ...
205  *   'k' = 107
206  *   'l' = 108
207  *   'm' = 109
208  *   ...
209  *   'z' = 122
210  *
211  */
212 void runTrackingAlgorithm()
```

```
{
  // Set up 'videoCapture' to take video from webcam
  if ( videoTrack )
  {
    videoCapture.open( 0 );
    videoCapture.set( CV_CAP_PROP_FRAME_WIDTH, VIDEO_WIDTH );
    videoCapture.set( CV_CAP_PROP_FRAME_HEIGHT, VIDEO_HEIGHT );
  }


  // Create main windows
  setUpMainWindow();
  imshow( mainWindowName, imread( "intro.png", CV_LOAD_IMAGE_COLOR ) );

  switch( waitKey( 0 ) )
  {
  case 99: // If letter 'c' is pressed
    colourTrack = true;
    setUpColourWindows();
    break;
  default:
  case 109: // If letter 'm' is pressed
    differenceTrack = true;
    setUpMotionWindows();
    break;
  }

  input = waitKey(10);

  // While escape key (code = 27) not pressed, wait 40ms each
  while ( input != 27 )
  {
    if ( colourTrack ) trackByColour();
    if ( differenceTrack ) trackByDifference();

    input = waitKey(10);

    // If you press p, print coordinates of spots
    if ( input == 112 ) printCoordinates = true;

    // Press w / s to cycle up / down through sets of images
    if ( input == 119 ) {
      imageSetIndex == imageSetIndexMax - 1 ? imageSetIndex = 0 : imageSetIndex++;
      imageIndex = 1;
    }
    if ( input == 115 )
    {
      imageSetIndex == 0 ? imageSetIndex = imageSetIndexMax - 1 : imageSetIndex--;
      imageIndex = 1;
    }

    // Press a / d to cycle left / right through set list of images
    if ( input == 97 )
      imageIndex == 0 ? imageIndex = imageIndexMax - 1 : imageIndex--;
    if ( input == 100 )
      imageIndex == imageIndexMax - 1 ? imageIndex = 0 : imageIndex++;

    // If r is pressed, toggle erode operations
    if ( input == 114 )
```

```
272       erodeFrame = !erodeFrame;
273
274     // If t is pressed, toggle dilate operations
275     if ( input == 116 )
276       dilateFrame = !dilateFrame;
277
278     // If b is pressed, toggle blurring
279     if ( input == 98 )
280       blurFrame = !blurFrame;
281
282     // If m is pressed, toggle hsv indication
283     if ( input == 109 )
284       showHSV = !showHSV;
285   }
286 }
287
288 /*
289  * This function will track light spots by looking for pixel differences between two
        images.
290  */
291 void trackByDifference()
292 {
293   // If focus is to track by images, load image with error catching.
294   if ( imageTrack )
295   {
296     frame = imread( imageNames[imageSetIndex][0], CV_LOAD_IMAGE_COLOR );
297     if (frame.cols == 0) {
298          cout << "Error reading file " << endl;
299     }
300     flip( frame, frame, 0 ); // Silly me took pictures upside down
301   }
302   else
303   {
304     videoCapture.read( frame );
305   }
306
307   // Read the next frame so we can compare the difference
308   // This assumes that enough time has passed so that the next frame has already
        been captured by the camera
309   if ( videoTrack )
310   {
311     videoCapture.read( nextFrame );
312   }
313   else if ( imageTrack )
314   {
315     nextFrame = imread( imageNames[imageSetIndex][imageIndex], CV_LOAD_IMAGE_COLOR )
        ;
316
317     flip( nextFrame, nextFrame, 0 ); // Silly me took pictures upside down
318   }
319
320   // Resize windows to fit on laptop screen
321   resize( frame, frame, Size(), 0.2f, 0.2f );
322   resize( nextFrame, nextFrame, frame.size() );
323
324
325   // Convert 'frame' to gray scale;
326   cvtColor( frame, frameGray, CV_RGB2GRAY );
327   // Convert the next frame to gray scale;
```

```
328   cvtColor( nextFrame, nextFrameGray, CV_RGB2GRAY );
329
330   // Get the absolute different between pixel values in the two images
331   absdiff( frameGray, nextFrameGray, differenceFrame );
332
333   // Threshold the difference image out to get clearer motion
334   threshold( differenceFrame, differenceThresholdFrame, thresholdSensitivity, 255,
        THRESH_BINARY );
335
336   // Blur image to get rid of noise
337   if ( blurFrame && blurStrength != 0 )
338     blurImage( &differenceThresholdFrame, &differenceThresholdFrame, 1, blurStrength
        );
339
340   // Erode and Dilate 'thresholdFrame' to get rid of noise
341   if ( erodeFrame && erodeSize != 0 )
342       erodeImage( &differenceThresholdFrame, &differenceThresholdFrame, 0, erodeSize
        );
343
344   if ( dilateFrame && dilateSize != 0 )
345     dilateImage( &differenceThresholdFrame, &differenceThresholdFrame, 0, dilateSize
        );
346
347   // Track object in real camera feed based on threshold pixels
348   if ( trackFrame )
349     trackThresholdPixels( differenceThresholdFrame );
350
351   // Show result
352   imshow( mainWindowName, nextFrame );
353   imshow( differenceWindowName, differenceFrame );
354   imshow( differenceThresholdWindowName, differenceThresholdFrame );
355 }
356
357 /*
358  *  This function will track light spots by finding areas of a particular colour in
       an image.
359  */
360 void trackByColour()
361 {
362   // Read frame from 'videoCapture' and put into 'frame'
363   if ( videoTrack )
364   {
365     videoCapture.read( frame );
366   }
367   else if ( imageTrack )
368   {
369     frame = imread( imageNames[imageSetIndex][imageIndex], CV_LOAD_IMAGE_COLOR );
370     if (frame.cols == 0) {
371       cout << "Error reading file " << endl;
372     }
373
374     flip( frame, frame, 0 ); // Silly me took pictures upside down
375   }
376
377   // Pictures too big for my laptop screen...
378   resize( frame, frame, Size(), 0.2f, 0.2f );
379
380   // Convert 'frame' to HSV colour scheme and put into new Matrix 'hsvFrame'
381   cvtColor( frame, hsvFrame, CV_BGR2HSV );
```

```
382
383   // Find pixels from a specific colour range from 'hsvFrame', set those to one and
        all others to
384   // zero, and put result in new Matrix 'thresholdFrame'
385   inRange( hsvFrame, Scalar(hMin, sMin, vMin), Scalar(hMax, sMax, vMax),
        thresholdFrame );
386
387   // Blur image to get rid of noise
388   if ( blurFrame && blurStrength != 0 )
389     blurImage( &thresholdFrame, &thresholdFrame, 1, blurStrength );
390
391   // Erode and Dilate 'thresholdFrame' to get rid of noise
392   if ( erodeFrame && erodeSize != 0 )
393       erodeImage( &thresholdFrame, &thresholdFrame, 0, erodeSize );
394
395   if ( dilateFrame && dilateSize != 0 )
396     dilateImage( &thresholdFrame, &thresholdFrame, 0, dilateSize );
397
398   // Track object in real camera feed based on threshold pixels
399   if ( trackFrame )
400     trackThresholdPixels( thresholdFrame );
401
402   // Show result
403   imshow( mainWindowName, frame );
404   imshow( hsvWindowName, hsvFrame );
405   imshow( thresholdWindowName, thresholdFrame );
406
407   if ( showHSV ) printHSV();
408 }
409
410 /*
411  * This function sets up and initializes all the windows and trackbars used to alter
        parameters or show images in the program, specifically
412  * those used in the colour finding algorithm.
413  */
414 void setUpColourWindows()
415 {
416   // Create Colour Windows
417   namedWindow( hsvWindowName );
418   namedWindow( thresholdWindowName );
419   namedWindow( trackbarWindowName );
420
421   // Add Trackbars
422   createTrackbar( "Hue Min", trackbarWindowName, &hMin, hMax );
423   createTrackbar( "Hue Max", trackbarWindowName, &hMax, hMax );
424   createTrackbar( "Sat Min", trackbarWindowName, &sMin, sMax );
425   createTrackbar( "Sat Max", trackbarWindowName, &sMax, sMax );
426   createTrackbar( "Val Min", trackbarWindowName, &vMin, vMax );
427   createTrackbar( "Val Max", trackbarWindowName, &vMax, vMax );
428
429   createTrackbar( "Erode Size", trackbarWindowName, &erodeSize, erodeMax, setOdd );
430   createTrackbar( "Dilate Size", trackbarWindowName, &dilateSize, dilateMax, setOdd
        );
431   createTrackbar( "Blur Strength", trackbarWindowName, &blurStrength, blurMax,
        setOdd );
432
433   setMouseCallback( mainWindowName, onMouse, 0 );
434 }
435
```

```
436 /*
437  * This function sets up and initializes the main window that displays the unaltered
        image and the trackbars used to alter tracking parameters.
438  */
439 void setUpMainWindow()
440 {
441   // Create Windows
442   namedWindow( mainWindowName );
443
444   // Add tracking trackbars
445   createTrackbar( "Max Number of Objects", mainWindowName, &maxNumberOfObjects, 200
        );
446   createTrackbar( "Min Object Area", mainWindowName, &objectAreaMin, objectAreaMax )
        ;
447   createTrackbar( "Max Object Area", mainWindowName, &objectAreaMax, objectAreaMax )
        ;
448 }
449
450 /*
451  * This function sets up and initializes all the windows and trackbars used to alter
        parameters or show images in the program, specifically
452  * those used in the difference finding algorithm.
453  */
454 void setUpMotionWindows()
455 {
456   // Create Colour Windows
457   namedWindow( trackbarWindowName );
458   namedWindow( differenceWindowName );
459   namedWindow( differenceThresholdWindowName );
460
461   // Add Trackbars
462   createTrackbar( "Erode Size", trackbarWindowName, &erodeSize, erodeMax, setOdd );
463   createTrackbar( "Dilate Size", trackbarWindowName, &dilateSize, dilateMax, setOdd
        );
464   createTrackbar( "Blur Strength", trackbarWindowName, &blurStrength, blurMax,
        setOdd );
465   createTrackbar( "Sensitivity", trackbarWindowName, &thresholdSensitivity,
        thresholdSensitivityMax );
466 }
467
468 /*
469  * Function to track the thresholded pixels ( all set to 1 ) in a given Matrix
470  */
471 void trackThresholdPixels( Mat threshFrame )
472 {
473   int x, y;
474   int objectsWithinSize;
475   Mat temp;
476   vector< vector<Point> > contours;
477   vector<Vec4i> contourHierarchy;
478
479   threshFrame.copyTo( temp );
480
481   // Get contours of pixels set to one in thresholded image.
482   findContours( temp, contours, contourHierarchy, CV_RETR_CCOMP,
        CV_CHAIN_APPROX_SIMPLE );
483
484   numberOfObjects = contourHierarchy.size();
485   objectsWithinSize = 0;
```

```
486
487   if ( printCoordinates ) cout << "---- Spot Coordinates ----" << endl;
488
489   // Assuming that the only objects left in 'thresholdFrame' are what we want, track
        them all.
490   if ( numberOfObjects > 0 && numberOfObjects < maxNumberOfObjects )
491   {
492     for ( int index = 0; index >= 0; index = contourHierarchy[index][0] )
493     {
494       Moments moment = moments((cv::Mat)contours[index]);
495       double area = moment.m00;
496
497       if ( area > objectAreaMin )
498       {
499         x = moment.m10/area;
500         y = moment.m01/area;
501
502         if ( colourTrack ) circle( frame, Point(x,y), 10, Scalar(0,255,0), 2);
503         if ( differenceTrack ) circle( nextFrame, Point(x,y), 10, Scalar(0,255,0),
      2);
504
505         objectsWithinSize++;
506
507         if ( printCoordinates ) cout << "\t" << intToString( x ) << ", " <<
      intToString( y ) << "\n";
508       }
509     }
510
511     // Display how many objects are being tracked
512     if ( colourTrack ) putText( frame, "Spots Found: " + intToString(
      objectsWithinSize ), Point(10,20), 1, 1, Scalar(0,255,0), 2);
513     if ( differenceTrack ) putText( nextFrame, "Spots Found: " + intToString(
      objectsWithinSize ), Point(10,20), 1, 1, Scalar(0,255,0), 2);
514
515   }
516
517   if ( printCoordinates ) printCoordinates = !printCoordinates;
518 }
519
520 /*
521  * Function to ensure the erosion and dilation size is always odd
522  */
523 void setOdd( int val, void *)
524 {
525   if ( val != 0 )
526   {
527     if ( !(erodeSize % 2 == 1) )
528     {
529       erodeSize = ( erodeSize > 1 ? erodeSize - 1 : 1 );
530     }
531
532     if ( !(dilateSize % 2 == 1) )
533     {
534       dilateSize = ( dilateSize > 1 ? dilateSize - 1 : 1 );
535     }
536
537     if ( !(blurStrength % 2 == 1) )
538     {
539       blurStrength = ( blurStrength > 1 ? blurStrength - 1 : 1 );
```

```
    }
  }
}

/*
 *  Function that is called on mouse click that will print out the hsv value of the
    pixel at the mouse position.
 */
void printHSV()
{
  Mat image;

  if ( colourTrack ) image = frame.clone();
  if ( differenceTrack ) image = nextFrame.clone();

  Vec3b rgb = image.at<Vec3b>(mouseY, mouseX);
  int B=rgb.val[0];
  int G=rgb.val[1];
  int R=rgb.val[2];

  Mat HSV;
  Mat RGB = image( Rect(mouseX, mouseY, 1, 1) );
  cvtColor( RGB, HSV, CV_BGR2HSV );

  Vec3b hsv = HSV.at<Vec3b>(0,0);
  int H = hsv.val[0];
  int S = hsv.val[1];
  int V = hsv.val[2];

  char name[30];
  sprintf(name,"B=%d",B);
  putText( image, name, Point(150,40), FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  sprintf(name,"G=%d",G);
  putText( image,name, Point(150,80) , FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  sprintf(name,"R=%d",R);
  putText( image,name, Point(150,120) , FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  sprintf(name,"H=%d",H);
  putText( image,name, Point(25,40) , FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  sprintf(name,"S=%d",S);
  putText( image,name, Point(25,80) , FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  sprintf(name,"V=%d",V);
  putText( image,name, Point(25,120) , FONT_HERSHEY_SIMPLEX, .7, Scalar(0,255,0) );

  imshow( mainWindowName, image );
}

/*
 *  Functions that defines what happens upon a mouse event.
 */
static void onMouse( int event, int x, int y, int f, void* )
{
  Mat image;
  if ( event == CV_EVENT_LBUTTONDOWN )
  {
```

```
598         mouseX = x;
599         mouseY = y;
600     }
601 }
```

### A.1.2 MorphOps.h

The header file 'MorphOps.h' is the header for the 'MorphOps.cpp' source file, and contains the declarations of the functions found in it.

```
/*
 * MorphOps.h
 *
 * Header file for a set of Morphological Operations that can be applied to images
 *
 *  Created on: 28 Jan 2015
 *      Author: Spencer Newton
 */

#include <opencv/cv.h>
#include <opencv/highgui.h>

#ifndef MORPHOPS_H_
#define MORPHOPS_H_

void dilateImage( cv::Mat*, cv::Mat*, int, int );
void erodeImage( cv::Mat*, cv::Mat*, int, int );
void blurImage( cv::Mat*, cv::Mat*, int, int );

#endif /* MORPHOPS_H_ */
```

### A.1.3 MorphOps.cpp

The source file 'MorphOps.cpp' is the file containing the implementations of the functions declared in 'MorphOps.h'. The functions contained within are the morphological operations used in the post-processing steps of the spot tracking algorithm, namely Erosion and Dilation.

```
/*
 * MorphOps.cpp
 *
 *  Source file containing implementations of Morphological operations to be used on images.
 *
 *  Created on: 28 Jan 2015
 *      Author: Spencer Newton
 */

#include "MorphOps.h"

using namespace cv;

// ================================== Variables ================================== //

int erosion_size = 2;
int dilation_size = 2;
int const max_elem = 2;
int const max_kernel_size = 21;

// ================================== End Variables ================================== //

/*
```

```
24  * This function will 'erode' and image and get rid of any noise. This will
        eliminate small high-intensity pixels, make dark
25  * areas larger and make bright areas smaller
26  */
27 void erodeImage( Mat *src, Mat *dest, int kernelType, int kernelSize = 2 )
28 {
29   int erosion_type;
30
31   if( kernelType == 0 ){ erosion_type = MORPH_RECT; }
32   else if( kernelType == 1 ){ erosion_type = MORPH_CROSS; }
33   else if( kernelType == 2) { erosion_type = MORPH_ELLIPSE; }
34
35   Mat element = getStructuringElement( erosion_type,
36                                        Size( 2*kernelSize + 1, 2*kernelSize+1 ),
37                                        Point( kernelSize, kernelSize ) );
38
39   // Apply the erosion operation
40   erode( *src, *dest, element );
41 }
42
43 /*
44  * This function will 'dilate' an image and amplify any bright areas. This will make
        bright areas larger and dark areas smaller.
45  */
46 void dilateImage( Mat *src, Mat *dest, int kernelType, int kernelSize = 2 )
47 {
48   int dilation_type;
49   if( kernelType == 0 ){ dilation_type = MORPH_RECT; }
50   else if( kernelType == 1 ){ dilation_type = MORPH_CROSS; }
51   else if( kernelType == 2) { dilation_type = MORPH_ELLIPSE; }
52
53   Mat element = getStructuringElement( dilation_type,
54                                        Size( 2*kernelSize + 1, 2*kernelSize+1 ),
55                                        Point( kernelSize, kernelSize ) );
56   // Apply the dilation operation
57   dilate( *src, *dest, element );
58 }
59
60 /*
61  * This function will blur and image by blending the value of each pixel based on
        the sum of each
62  */
63 void blurImage( Mat *src, Mat *dest, int blurType, int kernelSize = 2 )
64 {
65   /*
66    * 0 = Homogeneous
67    * 1 = Gaussian
68    * 2 = Median
69    */
70
71   switch ( blurType )
72   {
73   default:
74   case 0:
75     blur( *src, *dest, Size( kernelSize, kernelSize ), Point(-1,-1) );
76     break;
77   case 1:
78     GaussianBlur( *src, *dest, Size( kernelSize, kernelSize ), 0, 0 );
79     break;
```

```
80    case 2:
81      medianBlur( *src, *dest, kernelSize );
82      break;
83    }
84  }
```