2016

Robot Detection Using Gradient and Color Signatures

Megan Marie Maher
Bowdoin College, meganmaher1@gmail.com

Robot Detection Using Gradient and Color Signatures

An Honors Paper for the Department of Computer Science

By Megan Marie Maher

Bowdoin College, 2016

Contents

List of Figures

Acknowledgements

My deepest gratitude to Professor Eric Chown and Bill Silver for their invaluable help, expertise, and encouragement. This work would not have been possible without them.

Abstract

Tasks which are simple for a human can be some of the most challenging for a robot. Finding and classifying objects in an image is a complex computer vision problem that computer scientists are constantly working to solve. In the context of the RoboCup Standard Platform League (SPL) Competition, in which humanoid robots are programmed to autonomously play soccer, identifying other robots on the field is an example of this difficult computer vision problem. Without obstacle detection in RoboCup, the robotic soccer players are unable to smoothly move around the field and can be penalized for walking into another robot. This project aims to use gradient and color signatures to identify robots in an image as a novel approach to visual robot detection. The method, "Fastgrad", is presented and analyzed in the context of the Bowdoin College Northern Bites codebase and then compared to other common methods of robot detection in RoboCup SPL.

1 Introduction

RoboCup is an international robotics event in which teams of researchers program autonomous robots to compete in soccer matches, rescue missions, or home tasks. Bowdoin College's RoboCup team, the Northern Bites, competes in the Standard Platform League (SPL), where all participating teams use the humanoid Nao robot of Aldebaran Robotics.

Soccer, although simple for a human, is extremely difficult for a robot. When a robot moves around any given space, for each movement it must calculate the position of its joints and adjust in order to not fall over. It must take in information from its cameras and sensors and figure out what makes up its surroundings, and where it is located in space. The robot may be exploring unknown territory and must react on the spot to changes in its environment. However, since it is a robot, all of these things must be carefully thought out and programmed by a human before it begins its task.

This project explores robot detection as an extension of the new vision system the Northern Bites implemented last year. In gameplay, robot detection is absolutely crucial. A robot must efficiently and effectively navigate around the field while avoiding other robots. If unable to detect obstacles on the field, a robot may be physically blocked from moving around, incapable of seeing around obstructions. Furthermore, it may walk directly into a robot in its path and would most likely cost itself a penalty of 45 seconds, which is a significant portion of each 10 minute half. As the league moves further from wireless communication, robot detection becomes even more essential.

Although it is effortless for a human to identify a robot on the soccer field, this task is extremely complicated for a robot. Robots are mostly white with a few grey spots; in comparison to the white goal posts, the white net, the white field lines, the partially white ball, and any white object in the background, it is easy for a robot to mistake something else for a robot. Time constraints further complicate this task, as all code must be run 30 times every second, including the behavior system, localization, motion, and vision. Robot detection is only a small portion of what makes up a vision system, meaning any algorithm must be tremendously fast.

Specifically, there is around 1 ms free in the current vision system for the new algorithm. This is an upper bound, as any new addition to the vision system will take up some of this free 1 ms. Given the 1.6 GHz CPU of the Nao robot, there are around 1.6 million clock ticks every ms. Aldebaran's Nao robots have two cameras: one above their "eyes" and one below; these are referred to as the "top" and "bottom" cameras. Two images are processed every frame: one from each camera. Although the initial size of these images is 640 x 480 pixels, the top camera image is reduced to 320 x 240 pixels and the bottom image is reduced to 160 x 120 pixels, to minimize the number of pixels processed. Between the two images, this leads to an average of $\sim$17 instructions per pixel, assuming one instruction per clock tick. This is in comparison against other "real time" visual obstacle detection algorithms [8], which may run at 5 frames per second on one image of size 128 x 128 pixels, providing time for $\sim$20,000 instructions per pixel.

Previous methods of robot detection in our codebase have not proven to be adequate. These methods include the use of sonars and the calculation of when the robot's arms have been disturbed by an obstruction from their intended location. The most obvious improvement to robot detection is to move to visual detection. Although there are a few existing methods of visual robot detection, the proposed "Fastgrad" method works cohesively with the Northern Bites' vision infrastructure and explores a new approach to visual detection that uses gradient and color signatures.

2 Background

Obstacle detection in RoboCup SPL involves a suite of different detectors. These include the calculation of when the robots' arms have been displaced by an object from its intended position, the use of sonars, and visual robot detection.

2.1 Arm Disturbances

The Northern Bites detect an obstacle after a collision by determining if the robot's arms have been significantly moved. Since specific arm joint commands are sent to the robot in gameplay, it is known exactly where the arms should be. Therefore, when the current arm position is read back in, comparing this to the intended position determines if the arm has been displaced, and in what direction it has moved. However, a robot must be in direct contact with an obstruction in order for it to identify anything, limiting the detection to a very short range. Furthermore, if an obstacle is detected in this manner, the robot has already run into an obstacle; this should be avoided in the first place.

2.2 Sonars

Using sonars is another method that searches for potential obstructions in front of the robot. Aldebaran's Naos have two sonar units with two transmitters and two receivers: one on the left and one on the right side of a robot's center chest button. The sonars can detect between 0.2m and 0.8m for the NAO V5 robot or between 0.25m and 2.55m for the NAO V4 robot. This would seem to be an ideal form of short-range robot detection, as it gives a distance to an obstacle from both sonars, from which we can figure out where the obstacle is exactly. However, sonar readings are noisy and inconsistent, and there are irregularities in the sonars of the V5 robots where identical scenarios often produce inconsistent and faulty readings. This creates a need for a per-robot sonar configuration and the result is still noisy and subject to error. Additionally, sonars can only detect objects in a short-range, providing no information about other parts of the field.

2.3 Vision

There are two main approaches to visual robot detection that are implemented in RoboCup SPL. These methods, 1) using scan lines and 2) looking for disturbances in the field horizon, were created by the top two teams in the league, B-Human from the University of Bremen [6], and UNSW Australia from the University of New South Wales [3], respectively. Other teams in the league use variations of these two main approaches, or in some cases use the actual code from B-Human or UNSW Australia.

2.3.1 Scan Lines

B-Human's visual robot detection method [4, 5] looks for regions in the image which are both non-green and are larger than a field line. The algorithm is run after a number of other visual systems have already been processed. Most importantly, this includes finding the field horizon that is used in the robot detection algorithm.

To analyze an image, B-Human's algorithm only scans a small sample of points below the field horizon in the image to avoid wasting time looking at insignificant pixels. Specifically, points are examined at the intersection between parallel vertical lines and parallel horizontal

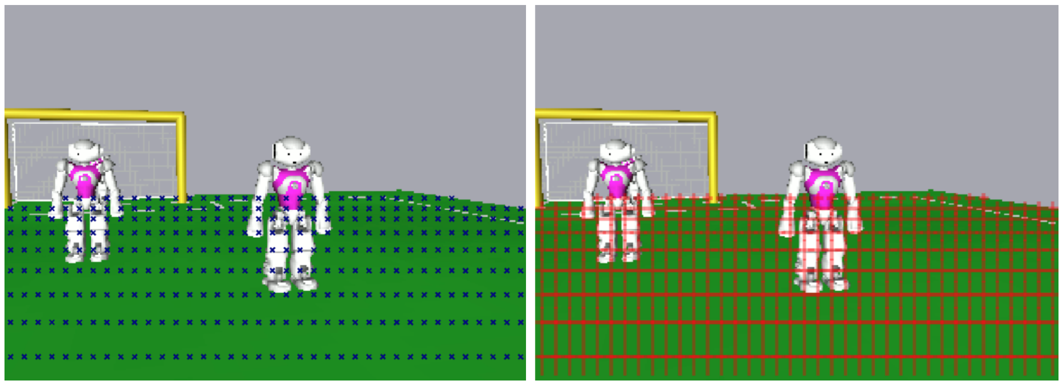


Figure 1: A sample set of pixels below the field horizon are analyzed, found along scan lines. Scanned pixels are shown in the first image [4], which are found at the intersection of the scan lines shown in the second image [4].

lines, where the space between vertical lines is equidistant and the spacing between the horizontal lines increases further down in the image (shown in Figure 1). The space between the horizontal lines reflects a fixed size in world coordinates, and corresponds to how they are projected into the image. The way in which field coordinates are transformed into world coordinates deals with Homography (see Appendix).

Pixels are analyzed starting where the field horizon meets each vertical scan line and moving downwards in the image. When points with mostly non-green pixels in their neighborhood are found in the image, they are marked for further examination. A pixel's neighborhood is made up of $2n$ pixels in each direction, where $n =$ the width of a field line. If there is more than one pixel on a vertical scan line that is surrounded by a mostly non-green neighborhood, the lowest of all these pixels is specially marked.

In this approach, it is initially assumed that anything that is both larger than a field line and mostly non-green is a robot. At this stage in the process, every vertical line with marked pixels could contain a robot. However, since there is little information about the size or shape of the robot, false positives are very likely. Thus, a series of filtering stages is necessary to rule out as many false positives as possible.

The first filtering stage checks the height in comparison to the width of the candidate robot. A scan to determine the bottom of the candidate moves downwards until areas of green are found. Similarly, to find the top, a scan moves upwards until either a patch of green is found or the horizon is reached. Any candidate whose height is less than its width at its lowest points is a false positive and is thrown out.

Next, robot hands which have been previously marked are removed. Since they are not directly on the field, it is impossible to use any knowledge about their location to calculate the distance to the robot. This is due to the way in which Homography functions (see Appendix). The robot's arms are found by examining the lowest marked points in each vertical scan line, and throwing out columns where the lowest marked pixel is significantly higher than the lowest marked pixels in the columns of the other vertical scan lines.

If part of the robot extends below the top image, the top and bottom images are stitched together. Once stitched, a sanity check is performed again to compare width and height in order to ensure that the candidate is not wider than it is tall. If a candidate passes all these stages, it is surrounded by a bounding box and classified as a robot.

B-Human's approach has proven to be very effective, finding robots in most frames.

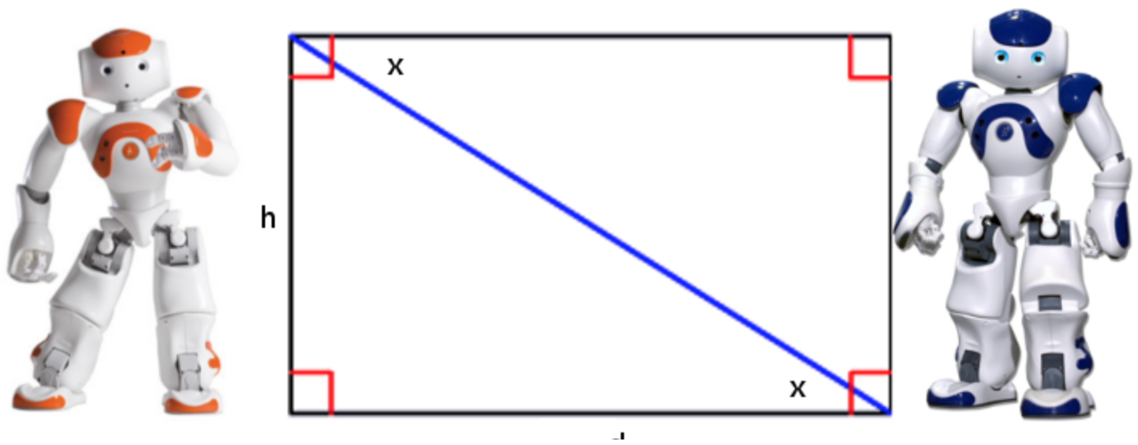


Figure 2: Trigonometry is used to calculate where the top of a robot is after finding the bottom. [1]

However, the method may have a hard time picking up robots in clumps, as clumps would be thrown out by the width vs. height sanity check. Clumps of robots are often the most important to identify, as they will be the largest obstruction on the field.

2.3.2 Disturbances in the Field Horizon

In 2014, UNSW Australia, previously known as rUNSWift, published a paper on a visual robot detection algorithm [1] that looks for disturbances in the field horizon, builds robot candidates based on where the field is obstructed, and then uses Bayesian machine learning to determine which robot candidates are truly robots.

Initially, the algorithm looks at the field edge to see if any points along the line are not a part of the green field. Moving across each column in the image, x pixels below the field edge are scanned (currently $x = 5$) and are each assigned a score according to their type: white = 3, field = -3, background = 1. The scoring gives certain kinds of pixels more weight and accounts for the imperfect conditions that often exist in RoboCup. If the total score in any column is greater than 0, this column is marked to have a potential obstruction.

For each marked column, the next step in the algorithm finds the bottom of the obstruction in the column. Iterating downwards, when an area of multiple successive non-green pixels is reached, the area is marked as the bottom of the obstacle. UNSW Australia uses trigonometry to determine the top, assuming all robots would be upright (see Figure 2). At this point, these obstructions, enclosed by a bounding box, are robot candidates.

To determine if the candidate is truly a robot, the next step uses Bayesian reasoning with a few features. First, the height of the robot in comparison to the width of the robot: rise over run. Similar to B-Human's logic, robots with a very steep or very small slope are most likely false positives, as robots are tall and flat to a certain degree. Second, the algorithm considers the percentage of the pixels within the box that are white, as robots are mostly white, except for the jersey. A large obstruction that is mostly non-white is not likely to be a robot. Last, the machine learning classifier looks at the sonar input and compares its distance estimate to the visually computed estimate. If the distances are radically different, the candidate is more likely to be a false positive.

With this approach, the Bayesian reasoning categorizes candidates into "robot" and "non-robot". In this way, most robots were correctly detected in the images. However, there are a considerable number of false positives and the algorithm has trouble detecting fallen robots or clumps of robots. False positives are often extremely harmful as they influence the robot's decisions in a negative manner, whereas false negatives are not quite as harmful in this situation because the robot simply has no information upon which to act. Additionally, as previously stated, detecting robots in clumps is important since they are a very large obstruction on the field.

UNSW Australia's reliance on the field edge to find robots can be troublesome in certain circumstances. First, finding the field edge is enough of a difficult task with its own sources of error. The Northern Bites use a similar field edge detection method and have found that classification of the field edge often alternates between near perfection and utter disaster. It is possible that green exists beyond the field, such as in a robot's jersey, a spectator's shirt, a green poster, etc., which causes problems when creating the field edge. Second, looking for mostly non-green pixels beneath the field horizon requires a good classification of green, which is tricky and changes with different lighting conditions. Last, for a robot who is close to the field edge, any error in the calculation of the field edge makes it likely that the robot will not be detected because it will not be "below the field edge."

3 Robot Detection Using Gradient and Color

This paper introduces a new approach to visual robot detection, Fastgrad, that aims to minimize some of the issues in the previous methods. As formerly mentioned, robots are almost entirely white. However, since there are a considerable number of white objects on the field – namely the field lines, goal posts, and ball – color signatures are not enough to uniquely identify a robot. A signature that can help distinguish a robot from the other white objects on the field is gradient.

Gradient is a vector, with a magnitude and direction, that measures the change in brightness. Brightness is a function of a surface and the angle at which light hits the surface. With a constant light source, as the surface moves, the angle at which the light is reflected changes and thus the brightness changes on the surface. Change in brightness gives us gradient. Since three-dimensional objects have curved surfaces, light will strike places on the object at different angles and will produce different brightness levels across the object. However, in a two-dimensional object, light will strike every part at the same angle, producing uniform brightness. When we have uniform brightness we have little or no gradient, but when we have non-uniform brightness, as on three-dimensional robots, we are guaranteed to have gradient. The fact that significantly more gradients appear on a robot's body than on field lines provides a strong signature for distinguishing robots from other objects in an image.

In this approach, a "white-gradient image" is created which weighs the importance of both white and gradient signatures simultaneously. The steps to produce the image include 1) estimating gradient, 2) employing fuzzy logic, 3) making use of information from a Hough line transform, then 4) moving a fixed-size box around every position in the white-gradient image to search for areas with high spatial density. Each of these stages is discussed in depth in the following sections.

3.1 Gradient Estimation

The top and bottom cameras of the Nao robot each produce a YUV image with a separate Y, U, and V value for each pixel in the image, where Y is the pixel's brightness and U and V are color components. The reason we have three values is because cameras aim to mimic the human eye, which also has three parameters.

Since gradient measures the change in brightness, the Y values in the original image are required to calculate the gradient at each pixel location. A Sobel Estimator [7] was used to estimate the x and y gradients, or partial derivatives, $\delta z/\delta x$ and $\delta z/\delta y$ at each pixel, for $z = f(x, y)$. A Sobel matrix has a 3 x 3 neighborhood, defined as follows for the x and y gradients, respectively:

$$m_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad m_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Consider a neighborhood of Y values around pixel e to be:

$$n = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

To calculate the x and y gradients at each pixel e :

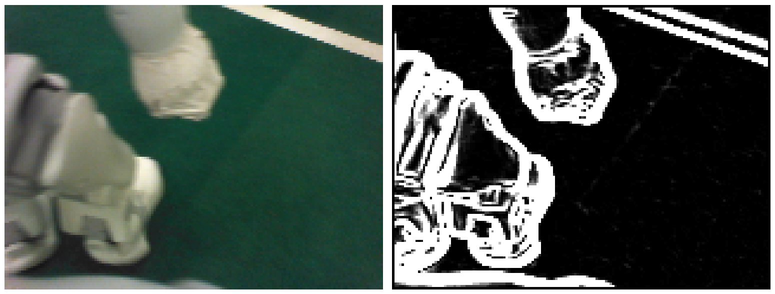


Figure 3: The original image is on the left and the gradient image is on the right. In the gradient image, the brightness of each pixel corresponds to the confidence that there is high gradient magnitude at that pixel location in the original image.

$$g_x = -1a + 0b + 1c - 2d + 0e + 2f - 1g + 0h + 1i$$

$$g_y = 1a + 2b + 1c + 0d + 0e + 0f - 1g - 2h - 1i$$

An alternative approach is to use a square kernel with weight matrices:

$$m_x = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}, \quad m_y = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

The Sobel estimator reduces noise significantly more than methods such as the square kernel, which is a notable strength given how noisy gradient can be. Although not as significant, gradient calculated with the Sobel estimator lies directly where the pixel e – the center of the weight matrix – is in the image. The gradient computed with the square kernel matrices produces a gradient value that exists between the four pixels in the neighborhood, which is not an addressable location in the image.

At this point, gradient values exist for every pixel location in the image. From here, fuzzy logic was employed to create a "gradient image".

3.2 Fuzzy Logic

The gradient image is a grayscale image where the value of each pixel corresponds to the confidence that there is high gradient magnitude at that location in the original image. To accomplish this, fuzzy logic was used to turn raw measurements into confidence values between 0 and 1.

The higher the gradient value of a pixel, the more important that pixel should be in the gradient image, up to a certain point. Conversely, the lower the gradient value, the less important that pixel should be in the gradient image, to a certain point. Fuzzy logic allows us to choose two thresholds, the upper and lower fuzzy thresholds, and then classify anything above the upper threshold with a confidence of 1 and anything below the lower threshold with a confidence of 0. Anything that falls between the two thresholds is scaled to the appropriate number between 0 and 1. In the gradient image, the solid white pixels were scored as 1 and the black pixels were scored as 0. Any brightness that was in between had a gradient value in between the two chosen fuzzy thresholds. The resulting image is shown in Figure 3.

Since robots have both gradient and white color signatures, it is important to highlight pixels in the image which reflect both qualities. In the Northern Bites codebase, a "white

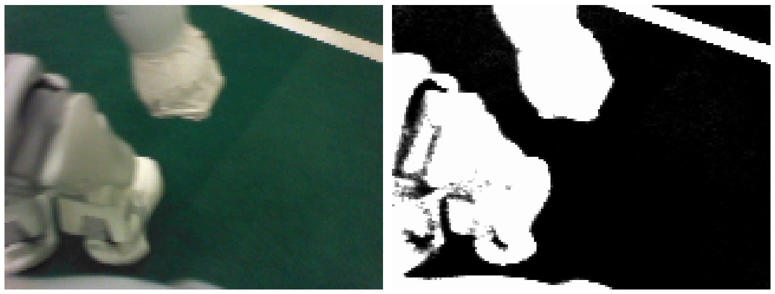


Figure 4: The original image is on the left and the white image is on the right. In the white image, the brightness of each pixel corresponds to the confidence that the pixel in the original image is white.

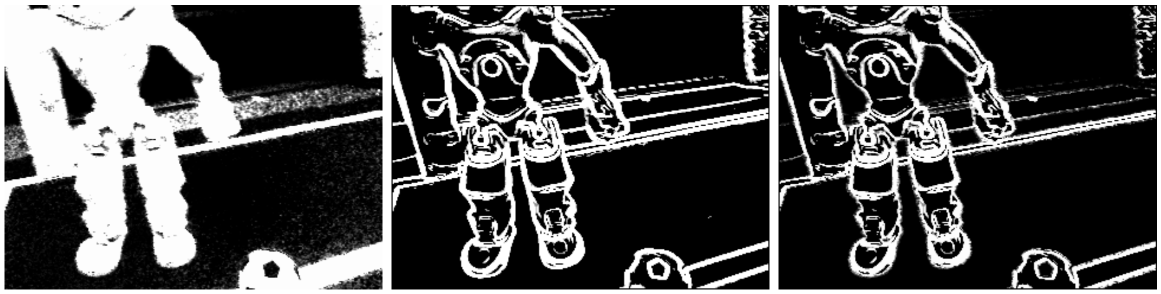


Figure 5: The combination of the white (left) and gradient (center) images produces a new white-gradient image (right), where the brightness of each pixel corresponds to the confidence that the pixel is part of a robot.

image" for the top and bottom cameras are separately created in each frame (see Figure 4). In this grayscale "white image", the brightness of each pixel corresponds to the confidence that the pixel is white. This was created with fuzzy logic, using a computed color (see Appendix).

The white images and the gradient images were merged to produce a new "white-gradient" image, where the brightness of each pixel corresponds to the confidence that the pixel has both a significant white color and high gradient magnitude. This refinement step rules out many false positives, as any pixel with only one of the robot signatures is most likely not part of a robot.

In fuzzy logic, when combining two fuzzy values, & ($*$) is defined to be the minimum value and | ($+$) is defined to be the maximum value. With these definitions, all boolean algebra properties still hold, including DeMorgan's Theorem, the associative property, the commutative property, etc. Since the goal is to produce an image whose brightness corresponds to both significant gradient and white values, the fuzzy & operator is used. This means choosing the brightness value in the white-gradient image to be the minimum of the two fuzzy brightness values at each pixel location in the two separate white and gradient images.

This resulting image (see Figure 5) highlights the two signatures that are most indicative of a robot. In other words, the brightness of each pixel in this image corresponds to the confidence that the pixel is part of a robot. There will always be a certain amount of error in these calculations, but there is an additional refinement step that uses field lines to rule out false positives in the image.

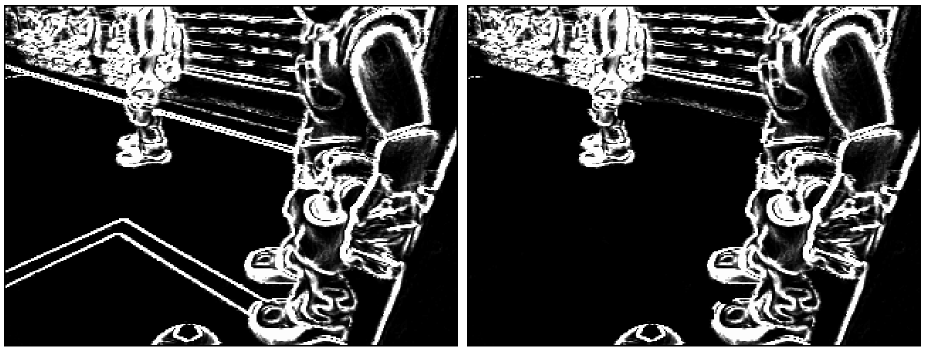


Figure 6: A refinement step that removes field lines from the white-gradient image.

3.3 Hough Line

The Hough Line transform is a widely used method of identifying straight lines in an image. The Northern Bites use this approach to very accurately detect lines every frame (see Appendix), producing a list of edges in the image that are part of known field lines.

Since field lines are completely white, they appear extremely bright in the white image. In addition, the field lines are significantly brighter than the green field beside them and therefore a large gradient is produced at the field lines' edges. Since the edges of the field lines appear very bright in both the white and gradient images, they are also bright in the merged white-gradient image.

Each refinement step in Fastgrad aims to decrease the number of potential false positives. Because the location of the field lines is calculated in each frame, it is simple to remove all pixels in the constructed white-gradient image that relate to the field lines. This step severely decreases the chance that part of a field line will be mistaken for a robot. Since there is never complete certainty when determining the location of field lines in the image, all pixels within 3 pixels of a field line are removed, or blackened, in the white-gradient image (see Figure 6).

3.4 Finding Areas with High Spatial Density

The white-gradient image's bright pixels are very white and have high gradient magnitude at that location in the original image. Areas that have high spatial density in this white-gradient grayscale image are the mostly likely to be a robot. To find these areas, a fixed-size box is moved to every possible position in the white-gradient image. At each location, the average brightness value contained by the box is computed and stored. Peak detection is then used to select the strongest candidates, whose average value is both above a certain threshold and higher than any of the neighboring candidates' average value.

Time complexity is always a huge concern in RoboCup. As previously mentioned, the entire robot detection system on both images must run in under 1 millisecond, with less than 17 instructions per processed pixel. However, moving a fixed-sized box around the image can be accomplished in linear time in the number of pixels in the image, independent of the size of the box.

A box size was chosen to reflect the average size of a robot in the image. Because the top and bottom cameras images are processed as different image sizes and show different parts of the robot, a different box size was chosen for the two images (see Figure 7).

The box algorithm begins with an array of w accumulators, for w = the width of the

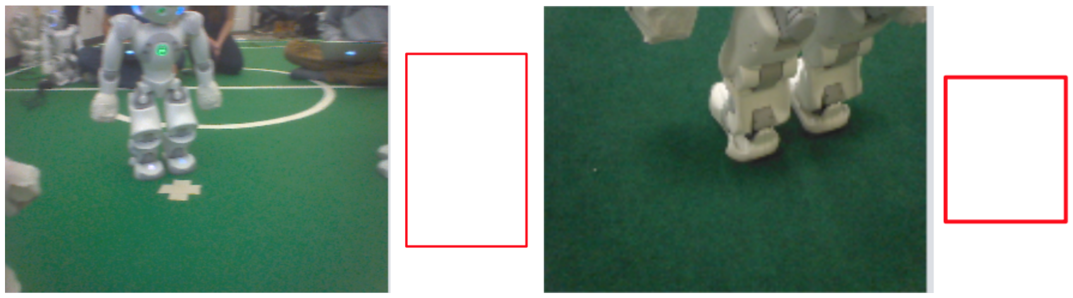


Figure 7: The top and bottom camera box sizes, respectively (two images not to scale relative to each other).

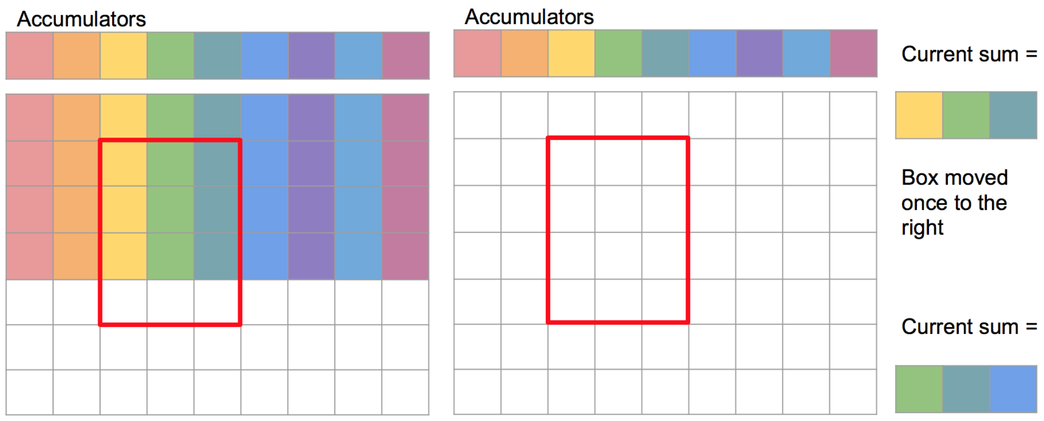


Figure 8: Left: For a sample image of size 7 x 9 pixels with a box of size 3 x 4, each index in the accumulator is the sum of the brightness values of the pixels that are highlighted with the same color. Right: As the box moves once to the right, the new accumulator is added and the old accumulator is subtracted.

image. Each accumulator is initially filled with the sum of the first h pixel brightness values in column x of the white-gradient image, where h is the height of the box and x is the current accumulator you are looking at (see Figure 8: Left).

Then, the first w accumulators are added to a "current sum", which reflects the sum of the brightness values at each pixel within a box, whose origin is in the upper left of the image. As the box is moved to the right, the next column's accumulator is added to the current sum and the previous box's first column accumulator is subtracted from the current sum. This continues until the end of the row (see Figure 8: Right).

For the next row and any subsequent row, each accumulator must be updated to reflect the sum of h pixels' brightness values in column x of the white-gradient image, starting at the current row and moving downwards. To do this, the brightness of the bottom right pixel of the new box is added to the accumulator corresponding to the rightmost column of the box, and the pixel brightness value directly above the top right pixel of the new box is subtracted from this accumulator (Figure 9).

Once the accumulator for the rightmost column of the new box has been updated, it can be added to the current sum. The accumulator for the leftmost column of the previous box, or the column to the left of the leftmost column in the new box, is then subtracted from the current sum. This is the same as what was formerly done in Figure 8: Right.

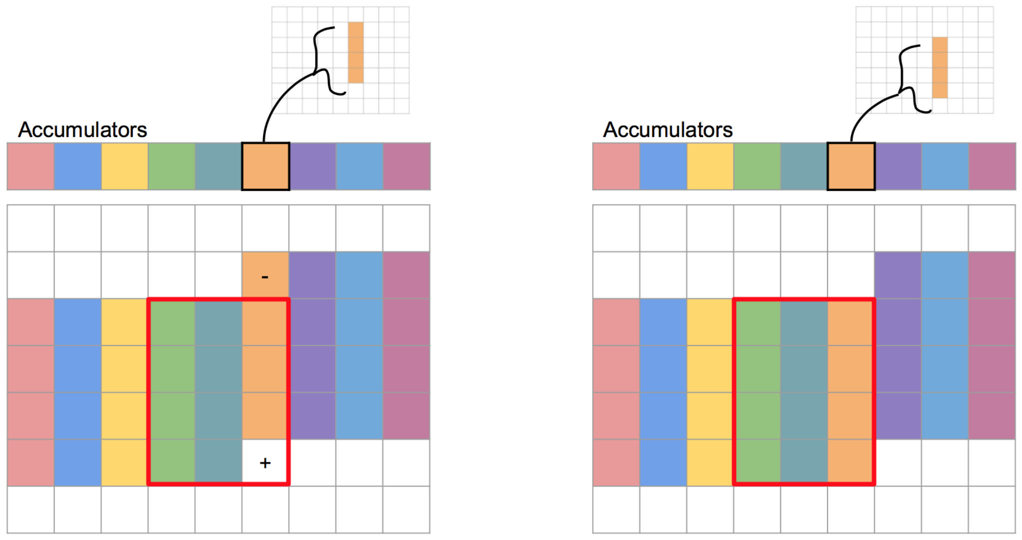


Figure 9: Before updating the current sum with the new accumulator, the new accumulator must be updated to reflect the rightmost column of pixels within the box.

In simpler terms, this algorithm follows the same repeated pattern:

```
for each row until row = height − box height
    for each column until column = width − box width
        // update current sum as follows for index (row, column)
        add bottom right pixel of new box to correct accumulator
        subtract pixel directly above top right pixel of new box from accum.
        add accumulator for right-most column in new box
        subtract accumulator for column before left-most column in box
```

The current sum gives us the total brightness within the entire bounding box. To get the average value, divide by the number of pixels enclosed by the box. This average pixel brightness value is stored for that specific box position.

Once the box has been moved to all possible locations in the image, we select box positions whose average brightness value is both above a certain threshold and is higher than all of its neighboring box's average brightness values. This is standard peak detection. Any box that meets these requirements is determined to be a robot.

3.5 Results and Evaluation

Overall, this method performed extremely well when both camera and color calibration were well tuned. Most robots were detected in both the top and bottom camera images, with a few exceptions due to the differences between the two images.

The bottom camera detection is extremely robust and is able to select correct robots in the image in difficult circumstances, such as when the image is mostly obstructed by a robot's own shoulder (see Figure 11 for a sample of images). The detection method has problems when robots are far enough away that their toes appear only in the very top of the bottom image. In this case, the box is too large to select any area containing a robot as having high spatial density. However, in this position, the robot is highly visible in the top camera, and we can rely on the top camera's detection to find the robot (Figure 10).

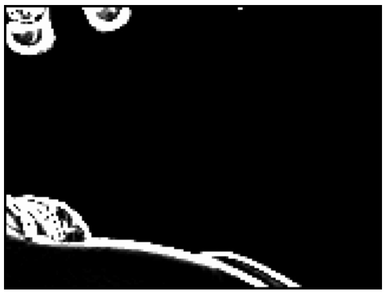


Figure 10: Bottom camera detection fails when the robot's feet are too high in the image. In this situation, the robot will likely be detected in the top camera.

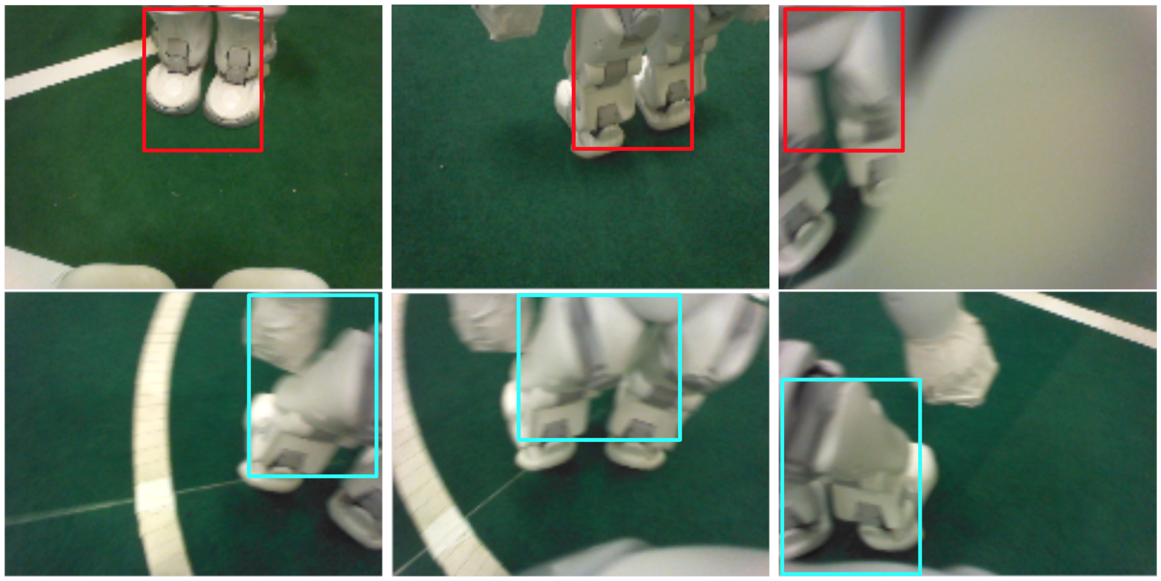


Figure 11: A collection of results from bottom camera robot detection. The blue boxes (bottom row) are a result of merging in around two red bounding boxes, on average.

In the top camera, the detection method is able to find robots in clumps. Clumps of robots often prove to be a challenging condition to classify in an image, but are some of the most important obstacles to detect since they are large obstructions. In UNSW Australia's robot detection method, a clump of robots would severely fail the rise-over-run test that is part of the Bayesian machine learning classifier. Similarly, B-Human's sanity check that compares the width and height of the candidate obstruction would immediately discard a clump of robots as a possible obstacle. However, since this newly-developed Fastgrad algorithm allows for multiple areas of high spatial density to be selected by the box algorithm, an entire clump of robots will be classified as an obstacle (see Figure 12 for detection of a clump, without merged boxes).

Though most circumstances resulted in correct robot detection, there are a few where the method does not detect robots in the top cameras. First, when robots are extremely close and only the chest and part of the head are in the image, the jersey of the robot is not picked up by the white-gradient image causing the box algorithm to fail (see Figure 12). However, when robots are this close, their feet will most likely be seen by the bottom camera. As the bottom camera robot detection is very robust, we can rely on it to detect

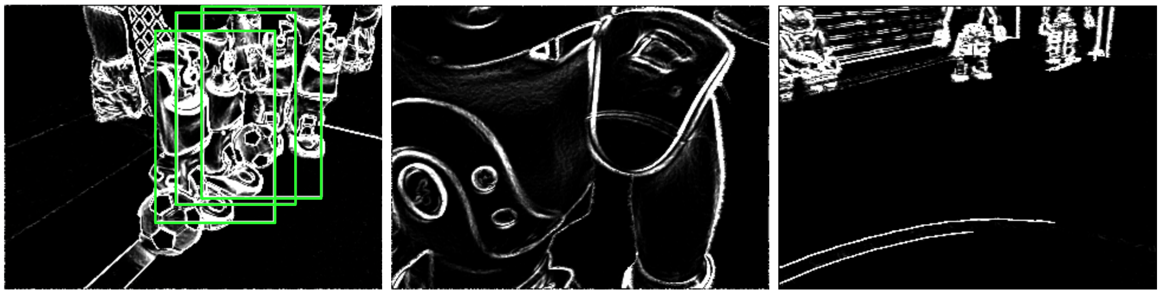


Figure 12: Unmerged detection of a small clump of robots (left), an undetected close robot (center), and undetected robots from a far distance (right).

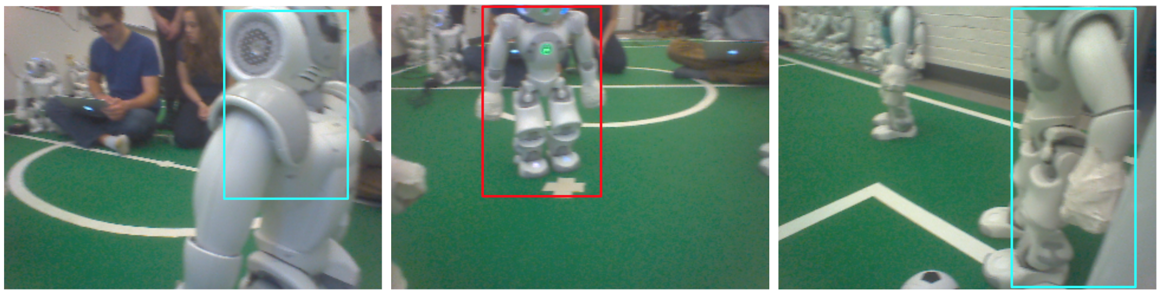


Figure 13: A collection of results from top camera robot detection. Merged boxes are shown in the left and right images and an unmerged box is shown in the center.

robots in these circumstances.

The top camera detection also fails when robots have fallen. Since the chosen box size is simply the average size of a robot in the top image, the box's orientation will not match that of a fallen robot. If the box were rotated 90 degrees, the fallen robot would most likely be detected. There has not yet been enough data to comment on the success of the algorithm overall when a second pass is made in the image with a horizontal box. The third situation when Fastgrad fails is when other robots are extremely far away (see Figure 12). In these cases, these other robots are especially small and the box is too big. However, detecting robots that are very far is not particularly important because no substantial decisions would be made based on the information anyway. In general, false positives are worse than false negatives in RoboCup because robots will act on incorrect information with false positives. Missing detection of a few robots in unusual circumstances will not hurt gameplay in a significant way, and it is not worthwhile to aim for absolute perfection.

4 Further Work

This method has taken steps towards developing successful robot detection, but there are many opportunities for further improvement. First, in adjusting the dimensions of the bounding box used to find areas of high spatial density in the white-gradient image. Second, in estimating the number of robots in a given clump. Last, in using jersey color to help with classification and to determine the team of a robot.

The bounding box's dimensions were chosen to match the average size of a robot in each image. However, as previously mentioned, this means that robots that are very far away or on the ground are not detected, as they do not conform to the dimensions of the bounding box. Moving forward, running the box algorithm with different box dimensions would be worth exploring.

In terms of finding robots on the ground, scans of the top image with the standard box rotated 90 degrees would highlight areas of high spatial density which are around the size of an average fallen robot. Many teams do not attempt robot detection for fallen robots because it is very easy to mistake field lines and balls for robots when relying on white color. However, the risk of mistaking field lines for robots is minimized in the *fastgrad* method due to the requirement of gradient signatures. Detecting robots on the ground is important to avoid walking into them, as when this happens, robots often end up piled on top of one another. Thus, exploring ways to find robots on the ground is a worthwhile task.

Adjusting the box size in general could allow for further detection of smaller robots in the background that currently are not large enough to be detected. At the moment, information about these smaller robots is not important as it is not used for anything in the codebase. However, if further versions of the code require information about small robots in the background, it would require adjusting the dimensions of the box in the box algorithm.

In this work, robot clumps are first characterized by multiple overlapping boxed areas with high spatial density and then they are merged into one large bounding box. Finding distinct robots or any object in a clump is a very complex computer vision problem with not many solutions, but doing so would provide very helpful information about tracking robots on the field. Although not very detailed, the large bounding box that this method provides can still provide information about the nature of the clump. If x robots are lined up facing the camera shoulder to shoulder and w is the width of one box, the bounding box would be xw. This provides an estimation of the minimum number of robots possible in the clump, as there must be at least x robots to create a box of size xw. However, there is no way to estimate the maximum number of robots in a clump, unless you use the maximum possible number of robots on the field as this number. For example, if x robots are lined up one behind the other in a straight line, where it appears from the front as if there is only one robot, only one robot is detected and the final bounding box will be of width w. There is no way to estimate how many robots are hiding behind each other.

Any jersey on a robot which is not white will be filtered out of the white-gradient image, as previously mentioned. Although the jersey is guaranteed to have gradient since it is stretched across a three-dimensional robot, it is not white and thus is not bright in the white image. This means the whole chest of the robot is guaranteed to be dark in the white-gradient image and the box algorithm will likely fail. Although there are not severe problems with this in the algorithm because the other parts of the robot were bright enough in the white-gradient image to compensate for the jersey pixels, determining jersey color would ease robot detection.

5 Conclusions

This project has enabled successful robot detection and has produced results that are usable in competition. It is a large improvement over the previous detection methods in the Northern Bites codebase, which included only the arm detection module that reports a collision after it has already occurred. The codebase now contains both reliable short range and long range robot detection algorithms which are very often able to identify obstacles and avoid collisions *before* they occur.

Using this new method of robot detection, we are less likely to have less collisions. Fewer collisions reduces the number of penalties as well as the number of times our robots fall down. This also means that our robots will be able to move down the field more easily and in a shorter amount of time, as they will not get stuck running into other robots. In addition, to avoid accidentally handing the ball off to the other team, information about obstacles allows the robots to aim kicks away from a potential opponent. These enhancements will likely mean smoother gameplay and a better performance from the team.

Overall, the robot detection method performed remarkably well and was able to find robots most of the time in the bottom camera as well as in key circumstances in the top camera. The goal was to create a system that significantly improved upon previous detection methods in the Northern Bites codebase as well as in other codebases in the league. Most importantly, the work demonstrated that robots can be uniquely identified with the use of only gradient and color signatures, and it has created a strong framework for visual robot detection with many opportunities for future development.

References

[1] Jaiden Ashmore. Robot detection using Bayesian machine learning, 2014. UNSW CSE RoboCup Report 20140831-Jaiden.Ashmore-RobotDetectionReport.pdf.

[2] Open CV. Hough Line Transform, 2014. Online; accessed December 12, 2015.

[3] Brad Hall, Sean Harris, Bernhard Hengst, Roger Liu, Kenneth Ng, Maurice Pagnucco, Luke Pearson, Claude Sammut, , and Peter Schmidt. RoboCup SPL 2015 Champion Team Paper, 2015. Available online: `http://www.cse.unsw.edu.au/opencms/export/sites/cse/about-us/help-resources/for-students/student-projects/robocup/reports/SPL2015ChampionTeamPaper.pdf`.

[4] Thomas Röfer, Tim Laue, Judith Müller, Michel Bartsch, Malte Jonas Batram, Arne Böckmann, Martin Böschen, Martin Kroker, Florian Maaß, Thomas Münder, Marcel Steinbeck, Andreas Stolpmann, Simon Taddiken, Alexis Tsogias, and Felix Wenk. B-Human Team Report and Code Release 2013, 2013. Only available online: `http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf`.

[5] Thomas Röfer, Tim Laue, Judith Müller, Dennis Schüthe, Arne Böckmann, Dana Jenett, Sebastian Koralewski, Florian Maaß, Elena Maier, Caren Siemer, Alexis Tsogias, and Jan-Bernd Vosteen. B-Human Team Report and Code Release 2014, 2014. Only available online: `http://www.b-human.de/downloads/publications/2014/CodeRelease2014.pdf`.

[6] Thomas Röfer, Tim Laue, Jesse Richter-Klug, Maik Schünemann, Jonas Stiensmeier, Andreas Stolpmann, Alexander Stöwing, and Felix Thielke. B-Human Team Report and Code Release 2015, 2015. Only available online: `http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf`.

[7] Irwin Sobel and Gary Feldman. A 3x3 Isotropic Gradient Operator for Image Processing. presented at the Stanford Artificial Intelligence Project (SAIL) in 1968.

[8] A. Talukder, S. Goldberg, L. Matthies, and A. Ansar. Real-time detection of moving objects in a dynamic scene from moving robotic vehicles. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 4, page 3718. IEEE, 2004.

[9] School of Computer Science University of Western Australia and Software Engineering. Computer Vision CITS4240, 2011. Online; accessed December 12, 2015.

[10] Wikipedia. Six degrees of freedom, 2016. Online; accessed December 12, 2015.

A Appendix

A.1 Color

Aldebaran's Nao humanoid robots have two cameras that produce raw YUV values, but robots work with color classes – such as white, green, orange – and so the raw values must be converted into a color class, like computed color. Computed color is an alternative to color table lookup that requires a calculation to convert from raw camera input instead of a table lookup. A color table lookup often means a cache miss, as the table has over two million entries and is too large to entirely fit in cache memory. In effect, performing a few linear calculations in parallel with computed color is generally faster than color table lookups.

Computed color is also simpler than color tables in terms of parameters. While each entry in the color table must be categorized before gameplay, computed color has a mere six parameters. This reduces hours of work that are normally required of color tables in a new environment. Conversely, computed colors are restricted to a color rectangle in the YUV space, whereas color tables allow for the creation of almost any color value.

A.2 Hough Line Transform

The Hough line transform is a crucial part of vision code that finds field lines in an image. The method looks at the Y value of the YUV input for each pixel and 1) calculates the gradient at each pixel, 2) maps all potential lines into Hough space, and 3) goes through a set of reduction steps to figure out what lines are field lines.

A.2.1 Step 1: Estimate Gradient

Looking at white solid line on a green solid background, there is a drastic change in brightness between the background (dark) and the line (bright). This change in brightness provides high gradient magnitude across the line's edge. The first step in the Hough transform is to estimate the gradient at each pixel in the image. This is done in the same manner previously described in this paper (Section III, Gradient Estimation), with the use of a standard Sobel Operator.

A.2.2 Step 2: Magnitude and Direction

Since every point in the image has both gradient magnitude and direction, there is a unique line that goes through each of these points. There are many ways to define a line. It can be represented by two points on the line, by the line's slope and a point it passes through, or even by the well-known equation $y = mx + b$. However, each of these methods either uses more than two parameters – such as two points on a line – or breaks in certain circumstances – such as $y = mx + b$ with an infinite slope.

In Hough space, there are two parameters that represent the line, in a way similar to polar coordinates. If the line is plotted, we define a line segment through the origin that is perpendicular to the plotted line. The closest point to the origin on the plotted line is where the line segment intersects the plotted line. The line is therefore defined by r, the length of the line segment, and θ, the angle the line segment makes with the x–axis (see Figure 14, Left).

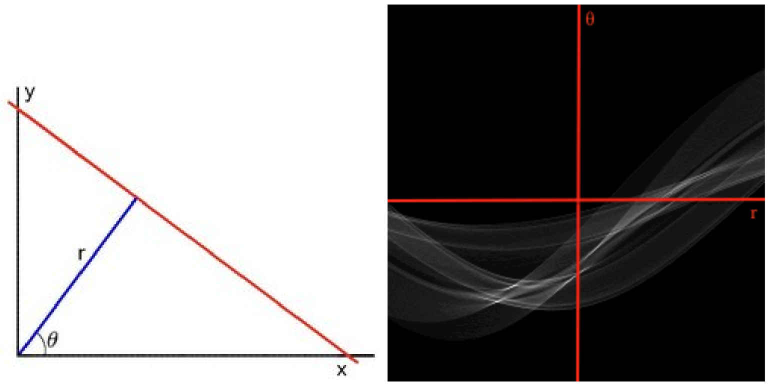


Figure 14: A Hough line is defined by two parameters: r and θ (Left [2]). A single point plotted in Hough Space represents a line (right [9]).

Now that there are two parameters, the line can be plotted by a single point in a two-dimensional space (r vs. θ). This two-dimensional space where points represent lines is called 'Hough Space' (see Figure 14, Right).

Hough space is treated as a grid of accumulators. The value of these accumulators represents the number of times a particular line has been found in the image. To build the accumulators, for each location in the image, the Hough space accumulator bin for the corresponding line is incremented. To account for noisy gradient calculation, bins are incremented that correspond to a few variations of the line. After incrementing the bins in Hough space, peak detection is used to select the bins which have a value greater than a certain threshold and a value greater than each of its neighboring bins. The selected bins correspond to lines that are the current field line candidates. To speed up processing, much of this computation is performed in assembly.

A.2.3 Step 3: Reduction

At this stage, there are an extraordinary amount of line candidates that are not field lines, and so it is necessary to go through a few sets of reduction steps to eliminate false positives: 1) pairing lines, 2) looking for correct lines, and 3) checking for lines that go from green to not-green.

First, pairing lines. Each field line has two edges, so two distinct line candidates that face each other and are relatively parallel are more likely to be part of a field line. The direction of a Hough line is taken from gradient direction, pointing from dark to light. To determine if lines are parallel, they first must be mapped into world coordinates via homography (see next section), and then checked for both parallel qualities and for a separating distance that is around the width of a field line.

The next step is to look eliminate lines that are too short, not straight enough, or don't contain enough points. Field lines are almost always at least a certain length in a RoboCup image, so if a line is too short, it can be discarded. To ensure the line is straight enough, a least-squares fit method is used. Edge points are points in the initial image with a high enough gradient magnitude, which were used to find Hough lines. When there are edge points that belong to more than one Hough line, they are assigned to the longest line possible. This reduces the number of points in smaller lines so they can be thrown out.

Finally, all field lines are white with a green background, so lines in the image that are

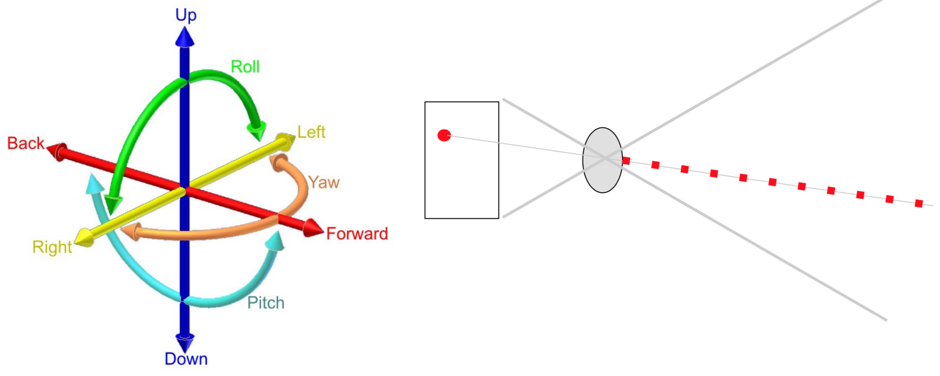


Figure 15: Left [10]: There are a total of six degrees of freedom: 3 translational and 3 rotational. Right: All points along the dotted red ray are mapped to the red point in the image. Reverse mapping, from image to world coordinates, is difficult because we no longer know where on the ray to map to.

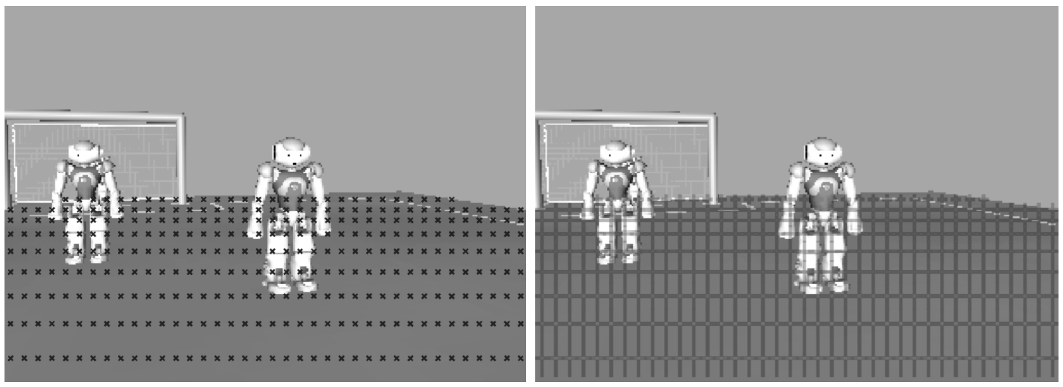
part of a field line will point from green to white. This last refinement step discards any line candidates that do not point from green to non-green. If any lines make it through all the reduction steps, they are classified as a field line.

A.3 Homography

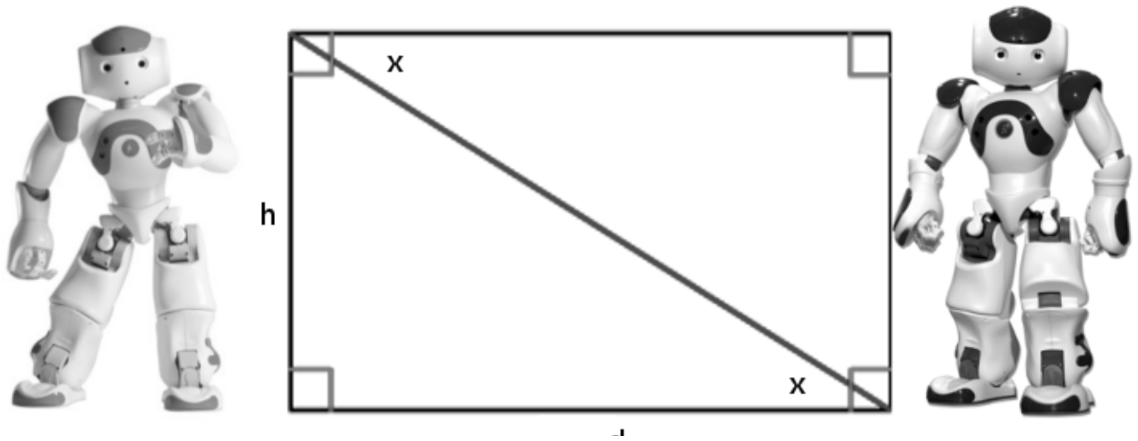
Homography is a perspective transformation between world coordinates and camera coordinates with both a linear and a non-linear component. Let's define the world's origin with coordinate system W and the camera's with coordinate system C. There is a translational and rotational difference between the origins of W and C, each with 3 degrees of freedom. This means a total of 6 degrees of freedom (see Figure 15, Left). The linear part of the transformation is the two-dimensional transform from world coordinates to camera coordinates, whereas the non-linear portion deals with the NAO camera lens model. Transformation matrices are used to perform any of these conversions between the two coordinate systems.

Homography is used in many circumstances to map between image coordinates and field coordinates. Mapping world to image coordinates is simple: there is only one place in the image that a point in the world can be mapped to. However, when mapping from image to world coordinates, there exists a whole ray of points that a pixel in the image could have come from in the world (see Figure 15, Right). Knowing where something is in world coordinates requires an additional piece of information. For example, when mapping field lines from image to world coordinates, it is known that the field lines will be on the ground. Knowing the height of the field lines – ground level – is enough to map to unique world coordinates.

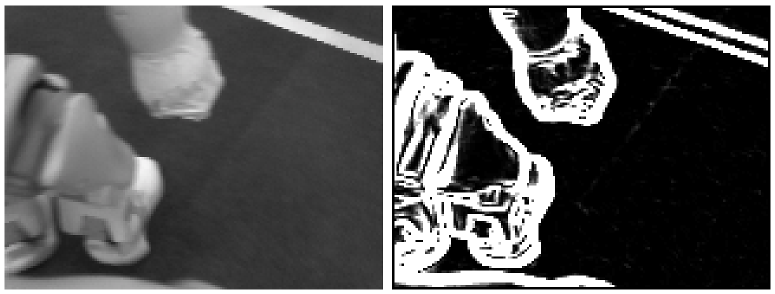
B Color Images in Black and White



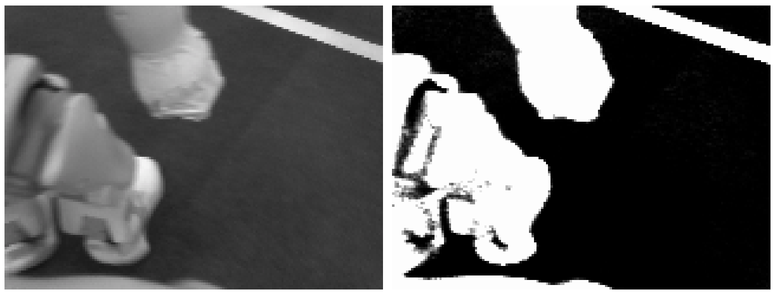
A sample set of pixels below the field horizon are analyzed, found along scan lines. Scanned pixels are shown in the first image [4], which are found at the intersection of the scan lines shown in the second image [4].



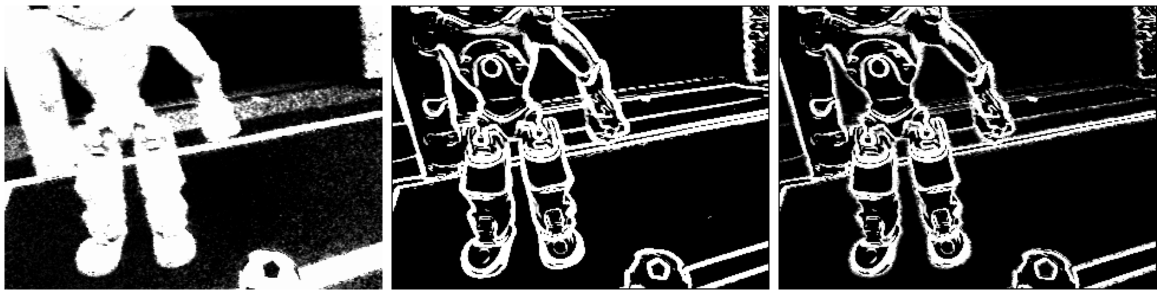
Trigonometry is used to calculate where the top of a robot is after finding the bottom. [1]



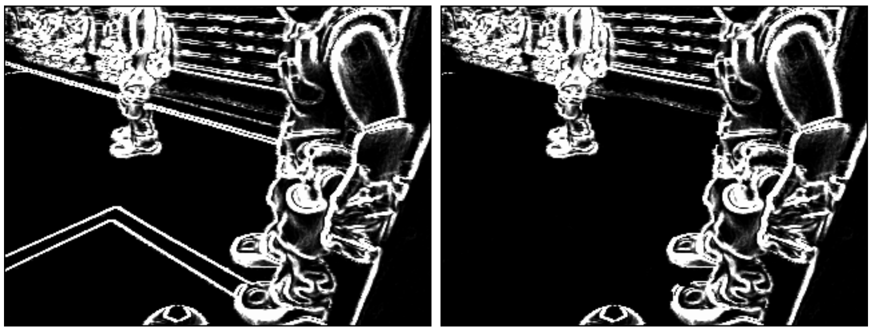
The original image is on the left and the gradient image is on the right. In the gradient image, the brightness of each pixel corresponds to the confidence that there is high gradient magnitude at that pixel location in the original image.



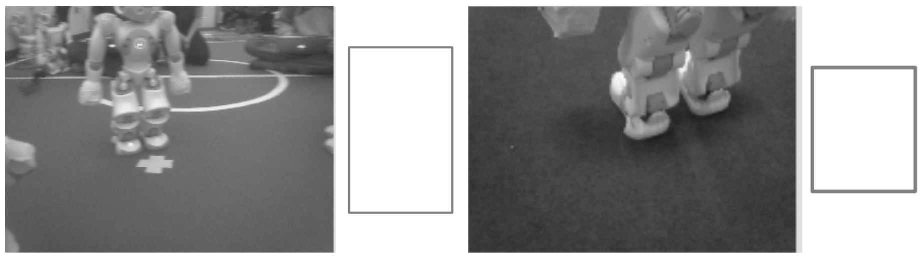
The original image is on the left and the white image is on the right. In the white image, the brightness of each pixel corresponds to the confidence that the pixel in the original image is white.



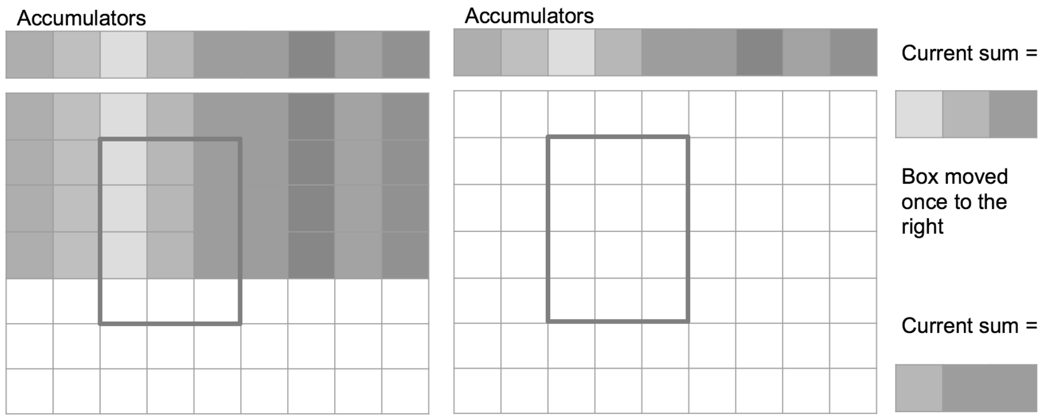
The combination of the white (left) and gradient (center) images produces a new white-gradient image (right), where the brightness of each pixel corresponds to the confidence that the pixel is part of a robot.



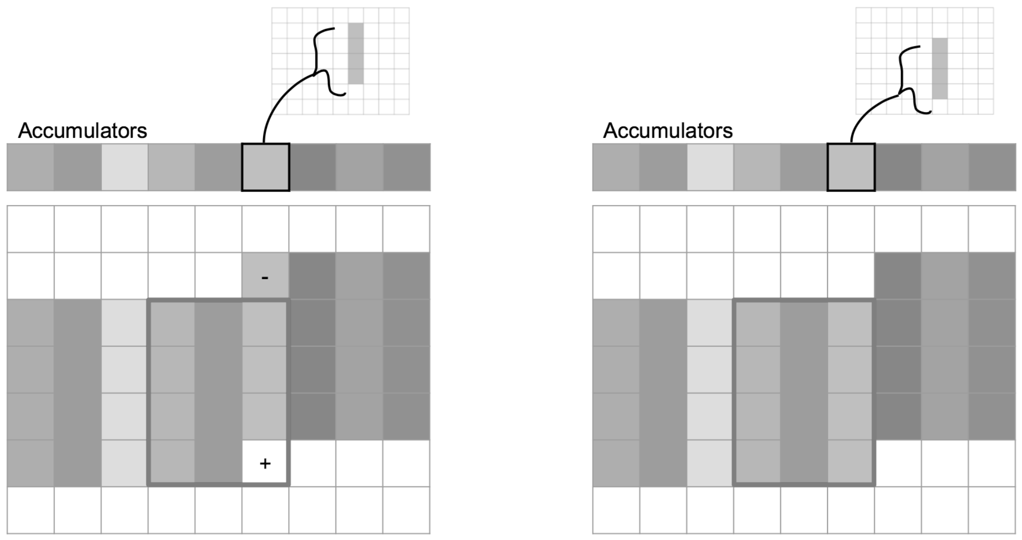
A refinement step that removes field lines from the white-gradient image.



The top and bottom camera box sizes, respectively (two images not to scale relative to each other).

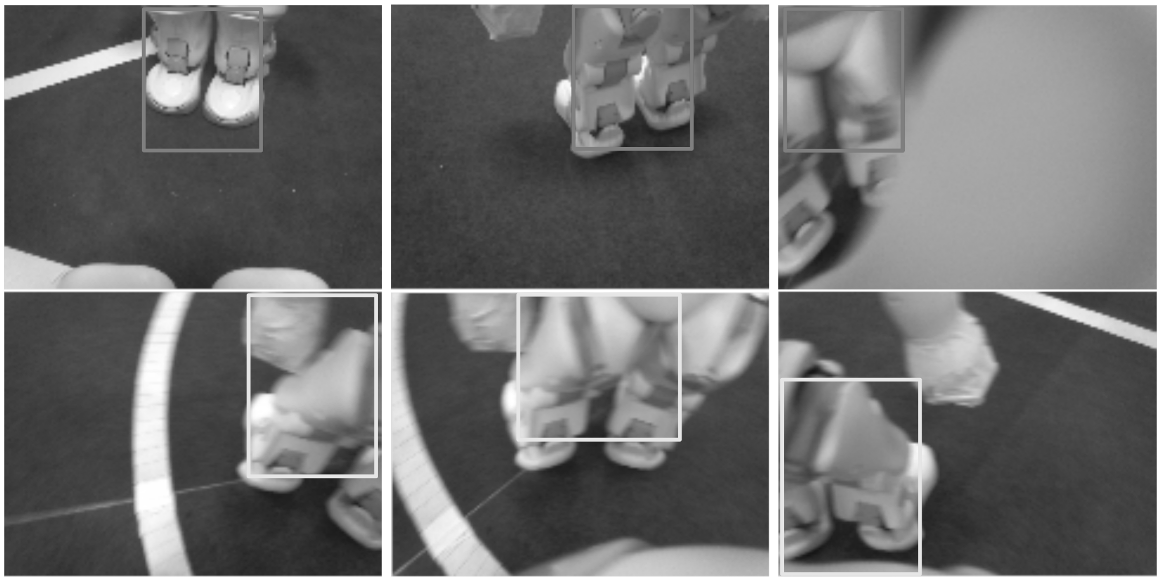


Left: For a sample image of size 7 x 9 pixels with a box of size 3 x 4, each index in the accumulator is the sum of the brightness values of the pixels that are highlighted with the same color. Right: As the box moves once to the right, the new accumulator is added and the old accumulator is subtracted.

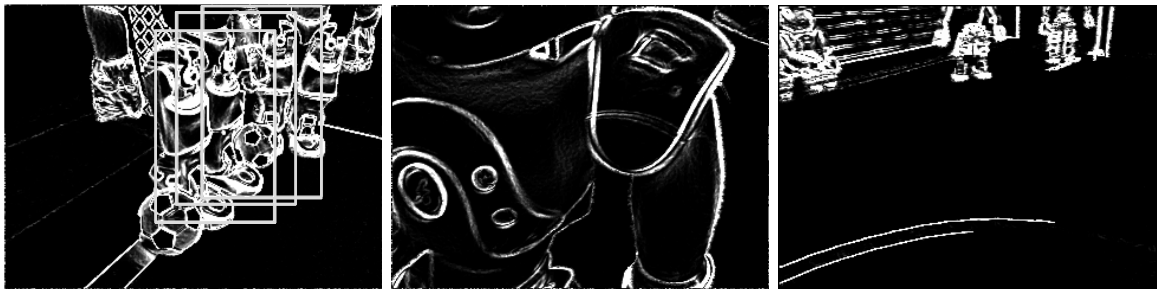


Before updating the current sum with the new accumulator, the new accumulator must be updated to reflect the rightmost column of pixels within the box.

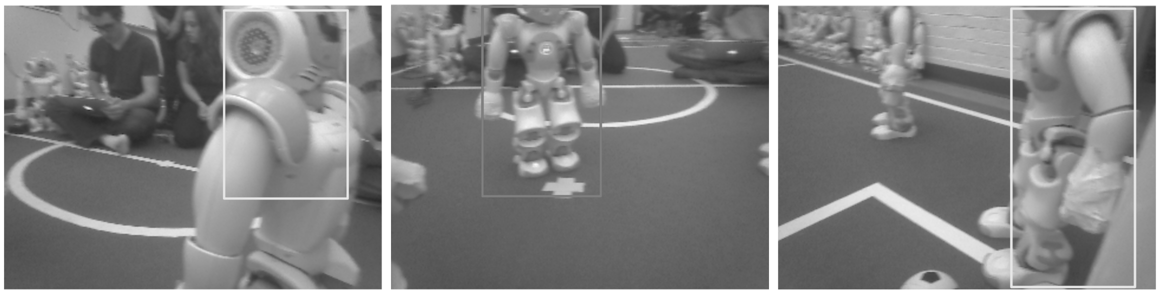
Bottom camera detection fails when the robot's feet are too high in the image. In this situation, the robot will likely be detected in the top camera.



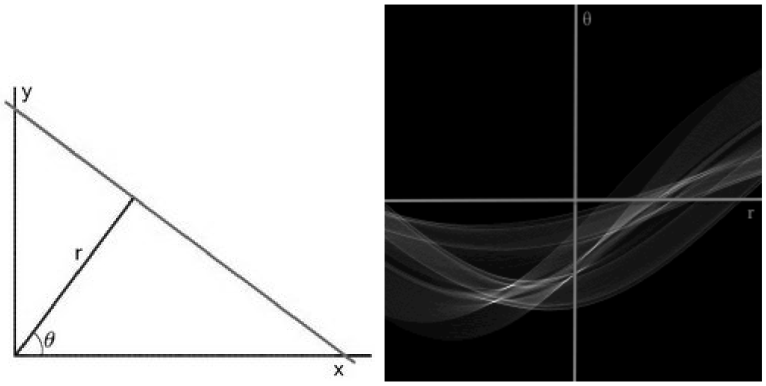
A collection of results from bottom camera robot detection. The blue boxes (bottom row) are a result of merging in around two red bounding boxes, on average.



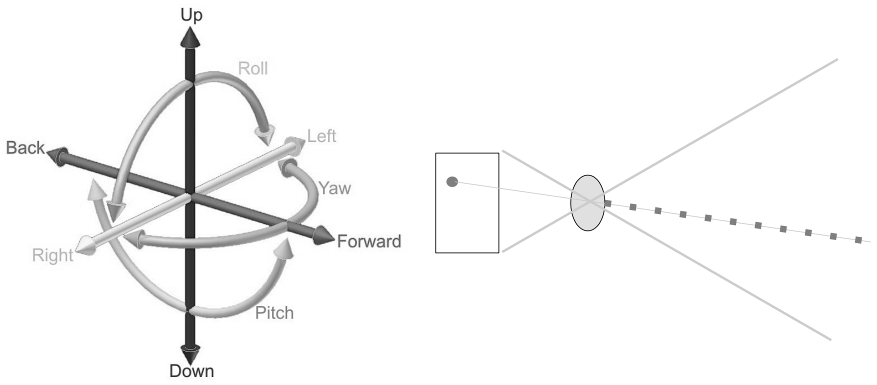
Unmerged detection of a small clump of robots (left), an undetected close robot (center), and undetected robots from a far distance (right).



A collection of results from top camera robot detection. Merged boxes are shown in the left and right images and an unmerged box is shown in the center.



A Hough line is defined by two parameters: r and θ (Left [2]). A single point plotted in Hough Space represents a line (right [9]).



Left [10]: There are a total of six degrees of freedom: 3 translational and 3 rotational. Right: All points along the dotted red ray are mapped to the red point in the image. Reverse mapping, from image to world coordinates, is difficult because we no longer know where on the ray to map to.