rUNSWift Team Report 2019

Jayen Ashar [1] **Kenji Brameld** [1] **Ethan R. Jones** [1] **Tripta Kaur** [1] **Liangde Li** [1] **Wentao Lu** [1] **Maurice Pagnucco** [1] **Claude Sammut** [1] **Qingbin Sheh** [1] **Peter Schmidt** [1] **Thomas Wells** [1] **Addo Wondo** [1] **Kelvin Yang** [1]

Abstract

RoboCup continues to inspire and motivate our research in artificial intelligence and robotics. The 2019 UNSW Sydney team (*rUNSWift*) consists of undergraduate students, Masters and PhD students, alumni, and supervisors. *rUNSWift* has a rich history, with involvement in RoboCup for over a decade in the Standard Platform League (SPL). We work in research areas that include computer vision, localisation, locomotion, machine learning, and layered hybrid architectures. Major developments in 2019 include an overhaul of the state estimation component of localisation, farther refinements to the vision system, and a range of improvements to behaviour and motion.

1. Introduction

Team *rUNSWift*, has been competing in the Standard Platform League (SPL) since 1999. Every year, we strive to improve the weakest aspects of our system and adapt it to new challenges presented by the committee through rule changes. In 2019, we focused on improving our localisation and behaviours, which we had identified as our primary weaknesses in 2018. With the improvements made over this year we were able to achieve 3rd place in the 2019 Robocup SPL.

2. Team Organisation and Development Methodologies

Consistent team organisation and procedures continue to play an important role in allowing us to improve our performance significantly each year.

The team maintained regular weekly meetings and testing routines that have been used in past years. We would meet once a week in person, with Google Hangouts for remote communication. At these meetings we would discuss team updates and technical direction for the next week. It also allowed team members to ask for assistance on difficulties they had experienced, fostering a culture of collaboration and support for other team members.

Our main form of communication is through Slack, a team-messaging platform, which allows messages to be sent out to specific members through the use of channels. This encouraged clean, organised team communication, allowing team members to only listen in to conversations of significance to them. Additionally our team maintains the code base in a git repository hosted on GitHub which enables effective collaboration with a large code base. We also utilise the GitHub wiki for internal documentation and GitHub issues to track ideas, issues and their completion.

We also maintained regular testing schedules. The simplest of these is a 'striker test', where a single robot and the ball are placed in set positions around the field, and the efficiency in scoring a goal is observed and compared against previous tests. This tests the integration of all our modules and improvements over the past week to ensure that changes are actually improving our overall soccer play. We also ran regular practice drills and matches involves multiple robots to evaluate team play, positioning, contention and obstacle avoidance.

3. Vision

The basic structure of the 2019 vision system remains the same as that of 2017 and 2018 (Bai et al., 2017; Brameld et al., 2018). The image is colour classified, (Section 3.1), regions of interest are detected based on this classification (Section 3.2) and finally regions of interest are classified into objects by detectors (Sections 3.3, 3.4 and 3.5). This layout is illustrated in Figure 1.

The key changes to our vision system in 2019 were:

- **Ball Detection Classifier** (Section 3.4).
- **New Robot Detector** (Section 3.5).
- **Field Features Classifier** (Section 3.5).

[1]School of Computer Science and Engineering, The University of New South Wales, Sydney, New South Wales, 2052.

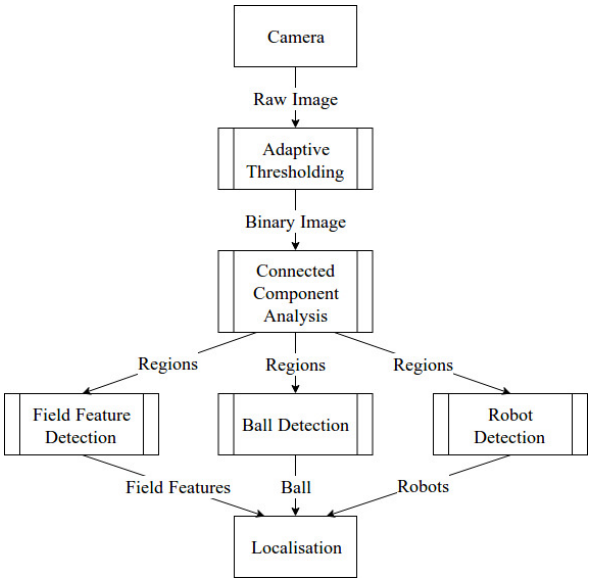

Figure 1. Overview of the 2018 vision system.

Figure 2. An example of the adaptive thresholding process. White pixels are considered light whilst green pixels are dark.

3.1. Adaptive Thresholding

In 2018 we converted the vision system to work with binary images to be more robust to varied lighting. This is done with adaptive thresholding, which is an effective method for calculating the threshold dynamically in scenes with uneven levels of brightness. This is a significant improvement on the static colour calibrated tables used by the team in previous years.

We utilise an efficient implementation of adaptive thresholding using the integral image (Bradley and Roth, 2007) to achieve real-time thresholding on the Nao. The algorithm determines the average y value (in the YUV422 image) of the pixels within a square window surrounding each pixel. If the value of the centre pixel is sufficiently lower than the average within the window it is set to black, otherwise it is set to white. This process can be performed efficiently, regardless of window size, with the integral image.

As this method does not rely on specialised colour tables the exposure of the image may be allowed to vary. This has allowed us to enable auto exposure on the camera, further improving our system's robustness to changes in lighting. Target exposure was brought down to minimise motion blur. Furthermore, manual colour classification of the environment before each game is no longer necessary, reducing the setup time. The main downside is the loss of colour information, previously used to determine the field boundary.

3.2. Regions of Interest

Currently, everything of interest on the RoboCup Soccer SPL field is primarily white. In the 2017 vision system, regions of interest (ROIs) were created by finding white areas in a colour calibrated image. Minimal changes was required to move to the binarised image in 2018. These ROIs are located using a connected component analysis algorithm (Rosenfeld and Pfaltz, 1966), with some modifications.

Since we just need the bounds of the regions, only the first pass of the algorithm is performed, recording the axis aligned extents of each group of pixels. When this is done connected groups can be quickly merged by taking the maximums of these bounds.

This algorithm tends to produce large groups containing most of the objects in the frame due to field lines connecting other white objects, such as the ball. As this is undesirable for the purposes of locating important objects the we split the scene into a grid, and prevents components from being connected across grid lines. Unfortunately, this may split useful objects like the ball so the system merges nearby groups where the edged of the bounding box have a sufficient ratio of white to not white pixels. This achieves bounding boxes that are reasonably likely to contain interesting objects, tightly bounded when those objects are not close to other white objects.

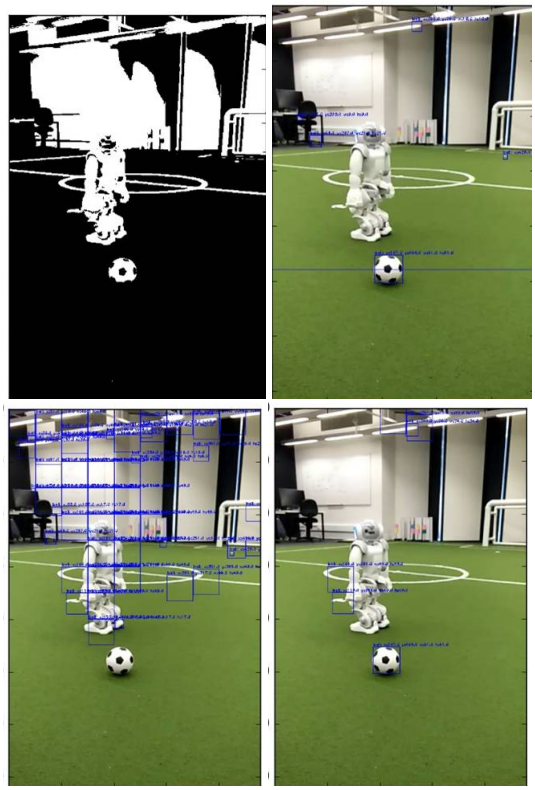

Figure 3. The region finding system. From left to right: The binary image. Regions generated without grid splitting. Regions generated with grid splitting, culled to show only ball like regions. Final regions generated with grid splitting and merging, culled to show only ball like regions.

3.3. Field Features

The field feature detector takes advantage of the ability of the ROI system to locate line segments. All lines of the field are white, such that they are detected and segmented by the ROI system 3.2. Some computationally efficient checks are performed on each region, allowing us to determine whether it contains a field feature. This is more efficient than the previous approach, which scanned the frame for candidate points and ran RANSAC on these points to locate field features. The detector is compatible with the binarisation of vision as it considers the light pixels as white (for lines) and dark pixels as green (for the field).

Our system detects regions that appear to contain line segments, corners, T-intersections and curved line segments. Curved line segments are combined together to form centre circle candidates, which are further checked for a line passing through the middle of the candidate circle to provide confirmation and orientation. If line segments intersect near the detected corner and T-intersection features these features can be confirmed, or even new, possibly occluded, ones detected. These lines and more distinct field features are passed to localisation to determine the position of the robot. This system is outlined in figure 4.

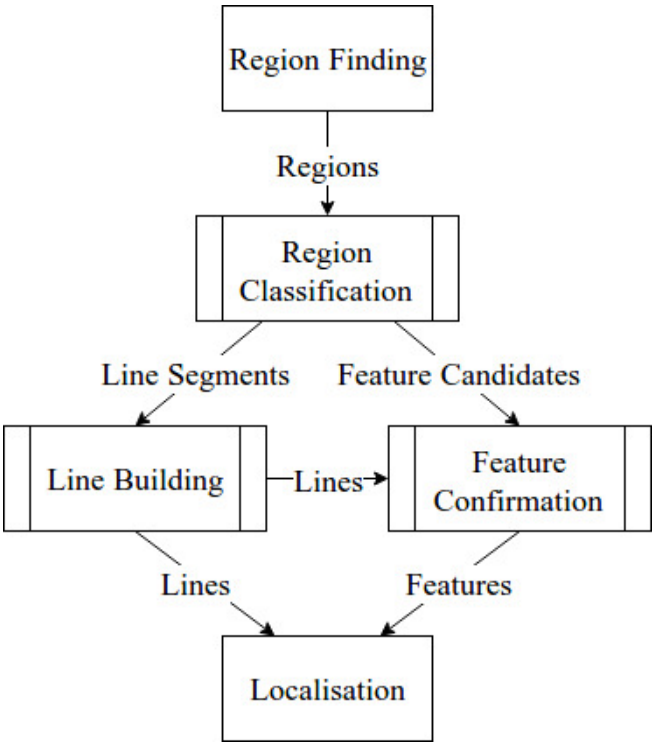

Figure 4. Overview of the field feature detector.

3.3.1. Region Expansion

There are times when a region does not contain a sufficient portion of a feature to identify it and accurately distinguish its direction. In such cases, we attempt to expand the region to obtain more information on the feature. For corners, this ensures the feature is fully included in the region. For curves, a larger curvature segment can be detected from the region. We can't expand a region arbitrarily, as in some situations, particularly near the goal box, features are located close to each other and expanding arbitrarily would put several features in a single region. We perform the expansion by combining some neighbouring regions, whilst trying to ensure that no other features have been included into the region. This additional step is effective for increasing the accuracy and consistency of detecting field features. The following pseudocode describes how this region expansion is performed:

3.3.2. Line Segments

The line classification system primarily examines the borders of regions. If exactly two white segments are found along the region boundary, they are assumed to be the ends of a line (Figure 5). We then check that the width of the two ends are approximately the correct size for a field line

```
For every region
    Count number of white pixel segments along the region edge
    Find all neighbouring regions
    For each neighbouring region
        Create region containing the original and neighbour region
        Count number of white edge sections in this region
        If number of white edge sections in original region AND combined region
                is the same
            Replace the original region with the combined region
```

based on an estimated projection from the robot's pose. We also check that the two ends are connected by following the edge of the white area. If these checks pass the system then attempts to determine if the segment is a straight line, a curve or a corner.

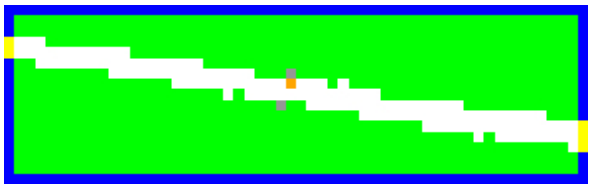

Figure 5. An example of a region classified as a line. Green areas are dark pixels, white areas are bright pixels. Blue pixels are dark areas on the border and yellow pixels are bright areas on the border. The orange pixel is the centre point between the line ends. Grey pixels are the extents of the line identified by the curve check.

3.3.3. Curve Segments

A region must be sufficiently large to be classified as a curve. Very small regions are always marked as lines. To check if a segment is a curve, the pixel half way between the two line ends is checked. If that pixel is green, this is a curve. If the pixel is white, the edges of the line around the centre point are found. If these are unevenly spaced from the centre point, the region is marked as a curve. If it is evenly spaced, it is considered a line. This is shown in Figures 5 and 6. If the segment passes these checks it is finally checked to see if it is merely curved or is an actual corner.

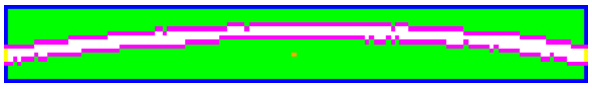

Figure 6. Curve detection on a curve. Most colours are as in Figure 5. As the centre point is not in the line there are no grey pixels. Pink pixels show the edges of the curve identified.

A region marked as curved must finally be checked to determine if it is actually a curved line or really a sharp corner. To do this the point along the edge of the white part of the line segment farthest from the line between the two ends is found as tips, the middle point of two tips is the intersection of this potential corner. If half of the edge points of white part is one a straight line, the region is marked as a corner (Figure 7). Otherwise it is left as a curve.

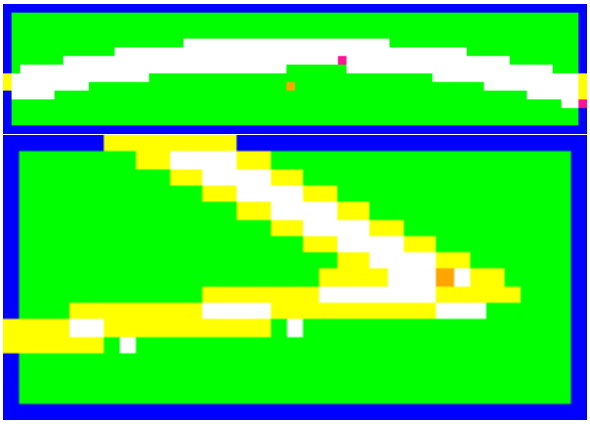

Figure 7. Examples of the corner check being performed on a curve (above) and corner (below). Here the orange pixels represent the "point" of the corner. The yellow lines represent the straight lines identified as composing the corner. The pink dots represent the best attempts to find the ends of those lines on the curve.

3.3.4. Corner Classifier

A region is marked as a corner candidate after the previous checks, it will be sent to a final classifier. This corner classifier is a specially designed semi-supervised module composed of a Gaussian Mixture Model (GMM) (Nielsen, 2012) and a simple classifier.

Two GMMs have defined a set of Gaussian models of both real corners and samples of non corner images that pass the heuristic checks. These unsupervised, generative models have data efficiency advantages (that is, they require fewer examples to learn) over many other machine learning methods, such as convolutional neural networks (Krizhevsky, Sutskever, and Hinton, 2012). When a new candidate is sent to the GMM, it will output the probability of each Gaussian in the model. Then, a simple Max A posterio algorithm is applied to determine the most probable classification by

calculating the probability of each mixture and picking the most probable as the final output.

3.3.5. Centre Circle Construction

Where curved line segments have been identified further checks are made to determine if the area can be confidently said to contain a centre circle. The possible centre circles corresponding to each curve segment are calculated. Line segments that lie on the edge of those circles are then identified. Three checks are made:

1. The total length of curved line segments along the circle edge must be above a threshold.

2. The total length of curved and straight line segments along the circle edge must be above a (higher) threshold.

3. There must be a sufficiently long merged line passing through close to the centre of the centre circle (i.e. the field centre line).

This is illustrated in Figure 8.

3.3.6. T Junctions

If there are exactly three white segments along the edge of a ROI, it is potentially a T intersection. As before, the white areas are checked to determine if they are roughly the correct size and connected to each other. If they are, the are further checked by ensuring the line between two of the ends is entirely white pixels (the top of the "T"). Finally the straight lines between these two ends and the other end (the base of the "T") must not be entirely white pixels. If all this is passed, the image is sent through a GMM, in a similar manner to the corner classifier (Section 3.3.4).

3.3.7. Feature Extrapolation

The final step performed by the system is to check that the corners and T intersections it has identified match the lines it has found. The system may also extrapolate corner and T intersection locations it did not find in regions. For every pair of (merged) lines the intersection point is identified. A quality value is then calculated for that intersection. If a corner or T intersection is detected nearby a large quality boost is added. The attributes of the lines forming the intersection are then examined. Longer lines with ends closer to the intersection add more quality. If one of the lines passes through the intersection quality is added to it being a T intersection, while corner quality is penalised (and vice versa). The line based intersection checks are illustrated in Figure 9.

3.3.8. Penalty Spot

Due to improvements to the state estimation system the goalie specific penalty spot detection used in 2018 was no longer required. This system was not used in 2019.

3.4. Ball Detector

Ball detection is a critical component of any robot soccer system. The 2016 transition to a black and white ball has made the task significantly more difficult. The ball can no longer be distinguished from other objects by its colour alone, instead requiring a more complex system that considers a candidate's appearance. This rendered the 2015 orange ball detector entirely ineffective. The 2017 ball detector adequately met this challenge primarily making use of the circular shape of the ball and its distinctive pattern. Despite this, the 2017 ball detector had a number of limitations and assumptions. The structure of the 2017 ball detector remained in 2018, with improvements made to address its limitations, increasing the reliability and accuracy. Finally, the 2019 system improved recall by relaxing many of the earlier checks in favour of making use of a new convolutional neural network in the final step, replacing the 2018 Gaussian mixture model.

Inside the ball detector, all regions are passed initially filtered through a ball candidate finder. The ROI finder at the vision level, described in 3.2 segments the white sections of the frame, however, these regions are not always a perfect crop of the ball. Namely, field lines, other robots and goal posts can be included in an ROI that also includes a ball. A tightly cropped and consistently scaled region of interest is desirable as it significantly simplifies the problem of ball pattern detection. Achieving a tight crop is the role of the ball candidate finder.

For "easy" cases cropping is done by Naive ROI (Section 3.4.1), while more complex regions require Blob ROI (Section 3.4.1). We also eliminate regions that are clearly not ball like due to size and shape. Scaling is performed in the region regeneration step (Section 3.4.2).

Next, region specific pre-processing (Section 3.4.2) is performed to adjust the image and make the major features of the ball candidate more salient. The white pixels are used to perform a circle fit, and we discard regions for which an adequate circle cannot be found (Section 3.4.2).

A ball candidate that reaches this stage undergoes two final heuristic checks to analyse the likelihood that it is a ball (Sections 3.4.3 and 3.4.3). Each of these checks give the candidate a score if passed. To be considered a ball a candidate must achieve a score above a certain score threshold.

3.4.1. Ball Candidate Generation

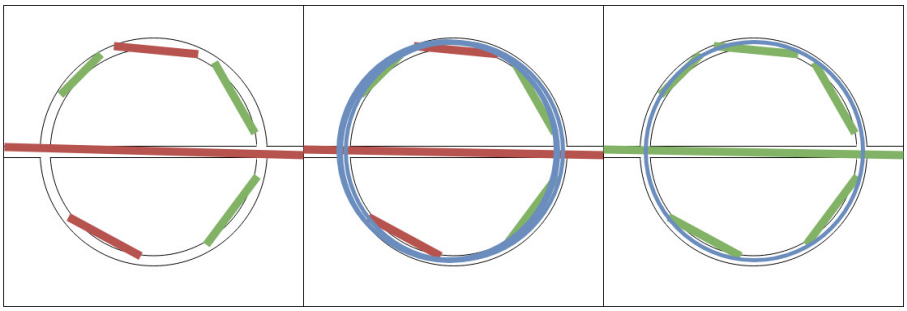

Figure 8. Basic process of constructing a centre circle. Left: The detected lines overlaid on the true centre circle, with green for curves and red for lines. Middle: All possible centre circles based on the curved lines. Right: One circle selected, with all the lines related to it in green.

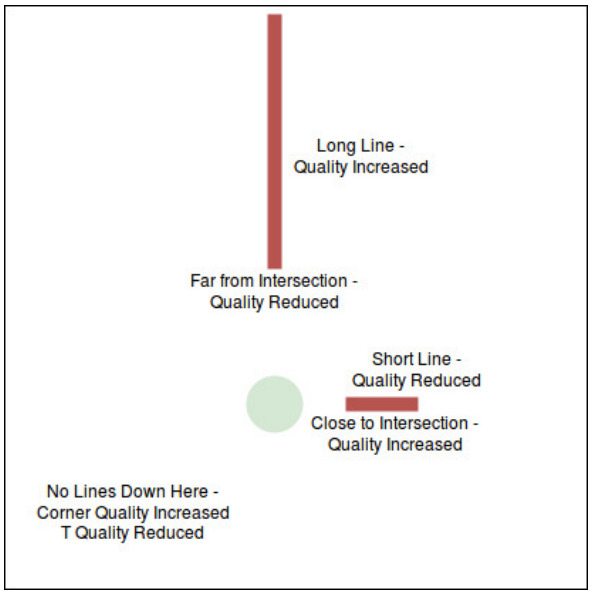

Figure 9. An example of the quality modifiers used by the intersection detector. Lines are shown in red and the intersection point is shown as a blue circle.

Naive ROI If a region is the correct size and shape for a ball, it is adjusted and considered as a candidate. The definition of correct size is calculated from the kinematics chain, given the ROI's position in the frame. The basic adjustments includes adding a small padding to the region in case the side or corner of the ball is lost and resizing the ROI to a fixed size. This is the most frequently occurring case in the game, where the ball is in the middle of the field with no obstructions.

Blob ROI When an ROI is larger than the expected size of a ball, there is a possibility of the ball being inside the region, but overlapping with other white objects. In this case, an attempt is made to create a tight region around the ball, by looking for the black blobs on the ball. A connected component analysis is run on the binarised region to locate all black blobs within the region. If multiple ball blobs are detected, they are compared to find the blob with the highest possibility of being from a ball. This is achieved by comparing the density of black within the bounding box around the blob. The density is determined by dividing the number of black pixels in the blob by the product of the width and height of the rectangular bounding box around the blob. The blob with the highest density is considerd the most likely to be a blob from the ball in the region. If the blob's width to height ratio is close to 1:1, a new ball candidate region is created around it. This region is then passed through the remainder of the detection pipeline as though it was a naive ROI. This helps us locate the ball when it is adjacent or overlapping with other white objects, such as when the ball is in front of a goal post or robot or on a line.

3.4.2. Preprocessing

ROI Specific Thresholding To sharpen the ball pattern, a second pass of the adaptive thresholding is run, local to the region. This is similar to the one described in Section 3.1, although the brightness parameter is determined dynamically for each region, based on its content. Specifically, if the region is in the top camera, the threshold percentage is one tenth of the average brightness of that ball candidate. For regions in the bottom camera, a single pre specified value is used, because there will be less variance in a single frame.

Circle Fit The ball candidate is refined further by using a circle fit. Candidate edge points are generated by scanning from the left, top, and right of the region to find the first transition from black to white in the binary image. A scan from the bottom is not conducted, as the shadow of the ball makes often makes its bottom edge difficult to identify. A circle fitting algorithm is then used to determine the existence of a circle, along with its centre and radius. The following describes the algorithm used for circle fitting, for a single region.

```
Create 2D array, size of the region for centre point voting
Create 1D array for radius voting
Find edge points by scanning region from the top, left and right of the region
For 100 iterations
    Randomly select three points
    Determine centre point and radius from points
    Vote on centre point
    Vote on radius
    IF max(centre point voting 2D array) > centre point voting threshold
            AND max(radius voting array) > radius voting threshold
        Found circle, with most-voted centre point and most-voted radius
```

Region Regeneration If a region has passed all the preceding checks it is considered a strong enough candidate to process further. The first step of this is 'zooming in' on the candidate region. If the region contains too few pixels when considered at the standard resolution of binary classification the classification is redone at higher resolution. The resolution is increased by the smallest factor of 2 that causes the region to contain the minimum number of pixels, or the maximum resolution of the Nao camera if needed. As a result, we are able to extend the range of the ball detection considerably, without having to process a high resolution image upfront in the stages of ROI generation 3.2.

3.4.3. Heuristic Checks

Triangle Check The regular shape of the pattern on the ball can be used to eliminate many non ball cases. To achieve this we check that the dark spots on the ball form a rough equilateral triangle. To detect these dark spots adaptive thresholding and connected component analysis is run specifically for the region. The parameters of these algorithms are, for this case, optimised to cleanly distinguish the black spots on the ball. Additionally, the circle fit is treated as a hard edge, preventing spots on the edge of the ball from being connected via the dark field. The black blobs found this way are checked to ensure they are of appropriate size and aspect ratio to be a ball spot, relative to the size of the circle.

The blobs that have passed these tests are then checked to determine if their centre points form an imperfect equilateral triangle. Each possible triangle is checked. As the triangle should be (roughly) equilateral this check can be easily performed by comparing the distances between each centre point. If all three distances are nearly the same they form a sufficiently equilateral triangle.

Machine Learning Classifier A major change this year is the replacement of the classifier for top camera ball detection. The class conditional Gaussian mixture Model (ccGMM) was replaced with a Convolutional Neural Network (CNN). The bottom camera still uses the ccGMM which was introduced in 2018(Brameld et al., 2018). Additionally a CNN has been used to improve our robot detection system.

Similar network structure is shared both by the ball and robot detector.

Table 1. Network Structure

Layer	Name	Output Size	Activation
0	Input	32*32*1	Linear
1	Conv	30*30*4	Relu
2	Max_pool	15*15*4	Linear
3	Conv	12*12*8	Relu
4	Max_pool	6*6*8	Linear
5	Flatten	288*1	Linear
6	FC1	8*1	Relu
7	FC2	2*1	Linear

Table 2 shows the network design for the ball detector. We split the training and inference into two separate parts. The network is trained using a Python3 program with Tensorflow(Abadi et al., 2015). The trained model's weights are saved in text format. These weight files are loaded when the robot starts. Inference on the robot is performed by a C++ program using tiny-dnn. Tiny-dnn is a header only dependency-free deep learning framework for C++14.

These two machine learning models have together improved our ability to recognise the ball. The bottom camera problem is simpler as the ball is usually more clearly visible, such that the cheaper ccGMM is sufficient. The more complex conditions in the top camera, such as additional potential false positives, varied lighting across the frame and the top half of the ball being brighter than the bottom half, benefit from the more expensive but reliable CNN. In practice detection range has been significantly increased without introducing additional false positives.

3.5. Robot Detector

As discussed in 2018's report, we use a series of heuristic checks along with a DBSCAN clustering to propose the robot regions(Brameld et al., 2018). This year we replace the random forest classifier with a Convolutional Neural Network (CNN). The main drawback of the previous random forest method is that it's performance largely depends on feature engineering, which is extremely challenging to develop. CNN do not require any feature selection, allowing the new model to achieve better generalisation. The design and training strategy is similar to the ball detector CNN which has been discussed in detail (Section 3.4.3).

4. State Estimation

One issue that arises in the SPL environment is the non-linearity between measurements from the robot's observation and the cartesian kalman filter. This issue is overcome by first converting the measurements from polar coordinates to cartesian coordinates before using it in the kalman filter. This method is used for both robot pose estimation and ball position and velocity estimation.

4.1. Robot Pose Estimation

Another issue that arises in the SPL environment is the ambiguity of field features (3.3) as there are multiple visually identical features that exist on different parts of the field (eg corners). To handle this, multiple modes (or hypotheses) are maintained and the possible robot position is calculated for each feature. The position that is most consistent with the robot's current pose estimate is used.

Each mode has a state vector (x, y, theta), covariance matrix and a weight. The mode with the highest weight is the current position estimate of the robot. The inputs used for the robot pose estimation are the odometry information from the robot and the distance, heading and orientation of all field features observed. The odometry is calculated from the motion of the robot's feet and the gyroscope reading. In the predict step, the odometry information is used to update the robot pose estimate for each mode. In the update step, the measurement coordinates (distance, heading and orientation) of field feature observations are converted to a set of possible robot poses in cartesian coordinates (x, y, theta). A linear kalman filter update is then applied to it. The weight of the mode is then scaled based on how well the current observation matches that particular mode. All modes are updated this way and finally the weights across modes are scaled, low weight modes deleted and similar modes merged.

4.2. Ball Position and Velocity Estimation

A converted measurement kalman filter is also used for the ball. A prediction of the ball's position and velocity is first made based on the change in time and the robot's odometry measurements. When a ball is observed by the robot, the position estimate is updated by first converting the non-linear polar measurements to cartesian coordinates and a linear update is then performed.

5. Motion

rUNSWift's motion is primarily based off the Hengst's walk generator (Hengst, 2014) developed in house and used since the 2014 RoboCup competition. Today, it is still robust choice for many teams and we have continued to improve it through several modifications. The kick motion, integrated directly into the walk generator for smooth transitions, was also developed around the same time. In 2018 competition improvements were made to both of these motions to improve the robots stability and recovery from disturbances. The motion of the kick was also modified to improve its dynamic reach, allowing the robot to powerfully strike balls in a larger area in front of the kicking foot.

In 2017 artificial grass fields were introduced, a surface that made balance recovery from disturbances more difficult. Additionally, the rough field surfaces at the 2017 RoboCup Nagoya competition farther complicated stable movement. These uneven surfaces introduce unexpected disturbances, causing instability. These factors, combined with an ageing and worn team of robots, influenced the decision to improve the stability of rUNSWift's walk generator. When a large destabilisation of the robot is detected by the gyro the robot will now react by moving its hip to compensate. As field construction proved similarly challenging in 2018 this development was tested in practice and proved itself a significant improvement to walk stability.

6. Behaviours

6.1. New Behaviour Structure

In 2018, we modified our behaviour architecture to improve dynamic role switching. From 2010, the higher level layers of rUNSWift's behaviours have been written in Python, allowing faster development. Our behaviours are modelled off a decision tree. Nodes are divided into two categories, roles and skills. Roles, such as "Goalie" and "Penalty Striker", define what the robot should be doing during a game. Skills are specific actions a robot has chosen to take.

Skills range from relatively high level actions, such as approaching the ball or walking to a global point on the field, to lower level actions, such as stand or crouch. This allows the low level skills to be inherited as components of several

higher level skills, which in turn are used by one or more roles.

State transitions can happen a frequently when a robot is involved in ball play. Under the old system, when transitions take place in high level skills, all child skills must be reinitialised. This process creates a computational overhead that reduced robot responsiveness, particularly during critical periods of play around the ball. To mitigate this we redesigned our system so that all objects are initialised at the start, and a reset routine is called on only the nodes of interest when a transition takes place.

7. Tooling

While they have no direct effect on gameplay, tools are critical to developing effective solutions to the problems found in robot soccer. Here the two primary tools used by rUNSWift are outlined.

7.1. Offnao

Offnao is a tool built in house that collects, saves, restores and streams data from Nao robots. It can record all the data from the robot's sensors, including either the colour classified or raw camera data. If given the raw camera data Offnao can also run the vision system offline. Offnao also collects information about the internal state of the robot, such as its beliefs about its position, team mate position and ball position, along with what the vision system has detected.

All this information can be displayed through various visualisation methods, including annotated versions of the robot's camera image and an overhead field view (Figure 10. Camera settings may also be changed through Offnao. Camera pose offsets can be calibrated to compensate for differences in the head mounting and looseness in the robot's joints. Debug logs recorded by the main code may be viewed. Finally, raw data from the non vision sensors, such as sonar and foot sensors, can be accessed.

This tool is also used to calibrate the camera pose and camera settings, by streaming live data from a robot that is running the code. We can tune these parameters in real time, before saving them to a config file for the robot to use.

7.1.1. Protocol

Protocol buffers are Google's language-neutral mechanism for [de]serializing structured data - think JSON, but binary. We define how we want our data to be structured once (like a JSON schema), then we use the generated source code to write and read our structured data to and from the network and disk.

Because it is JSON-like, it is backward and forward compatible across different versions and branches of our tools. Version 2.8 of Softbank's NAOqi OS and cross toolchain include protocol buffers version 2.6.1, and we have compiled protocol buffers for Aldebaran's NAOqi OS 2.1.

The generated C++ API for protocol buffers attempts to consume an entire stream during deserialization. For this reason, our protocol includes a 32-bit little-endian integer before each protocol buffer "Blackboard" object indicating the size of the object, for deserialization over the network and in .bbd2 (BlackBoard Dump v2) files. .ofn2 (OFfNao v2) files contain exactly one "NaoData" object.

7.2. Vatnao

As part of our 2016-17 Vision System changes we developed a tool to assist in development and debugging. The tool, Vatnao, runs the vision system on a local machine using logs recorded on the robot. It allows the developer to examine variables, annotate images based on how Vision processes the raw frames, and even adjust configuration in real time. We used it heavily as part of development of the ball detector to speed up the development cycle, reducing testing time, quickly identifying problem cases and allowing us to quickly evaluate the effects of tweaks and changes to our code base.

Vatnao has also allowed us to break down the components of each detector. For example, in the case of the ball detector, each heuristic can be broken down and displayed separately to determine optimal values for the checks. The tool also handles other modules of vision, such as field feature and robot detection. In 2018, we have continued to develop and use Vatnao to improve vision as we port more of the vision system to use regions of interest 3.2.

8. Whistle Challenge

The whistle challenge was a research exercise designed to develop different approaches to sound localisation. This is a technique used to identify where a particular sound emanated from in terms of some positioning metric, such as cartesian coordinates. By knowing how far the sound was relative to a position and if it existed within a predefined boundary, various applications can be realised. A typical example that is often brought up is knowing whether or not a whistle was blown within the current playing field or another alongside it. Obviously the competitors want their robots to react to a whistle inside their field, but they also want them to ignore those on other fields, since randomly stopping during a game is rather inefficient.

Each competitor was able to use between one and five NAO humanoid robots during the challenge, with the competitor who used the least being favoured in a tie-break scenario. A standard 9m x 6m soccer field was used for the challenge wherein the selected NAO robots were placed in predefined

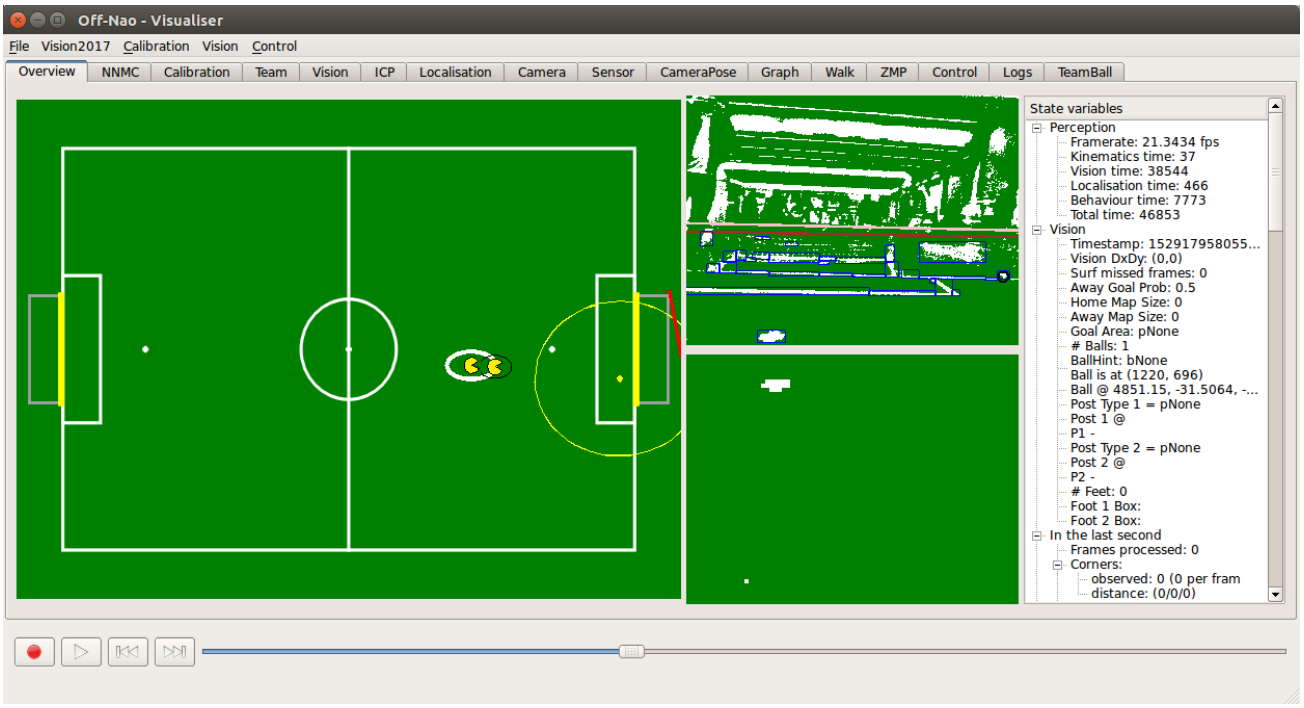

Figure 10. The primary Offnao display. The yellow dot is the current ball position belief. The little pac-man figure is the robot's position belief. Blue boxes are detected regions. The black circle is a detected ball.

positions and angles. If a competitor chose less than five robots, then they could also select which positions they preferred. There were no requirements pertaining to NAO hardware or software version, meaning that NAO robots from prior generations, such as the V3 or V4 variants, were permitted. Hardware modifications or additions were banned as to ensure a level playing field and promote the main purpose of the challenge.

The field specifications, robot positions and robot angles were defined two days before the challenge began to allow teams to properly incorporate them. Figure 11 provides an illustration of the field with these positions and angles included to allow a better understanding of such a setup.

Each robot had to be in a penalised state before being placed on the field where a penalised state meant that the robots were not performing any related processing. Once the robots were placed, their chest button was pressed exactly once and the whistle challenge code was executed. Each robot was not allowed to move during the challenge, they had to remain completely stationary with the exception of the head, likely because this housed the directional microphones.

After positioning, a person involved with running the challenge would navigate to eight different positions and blow a whistle. These positions were situated both in and outside of the playing field with no obvious correlation. The robots need to hear this whistle, generate a corresponding cartesian

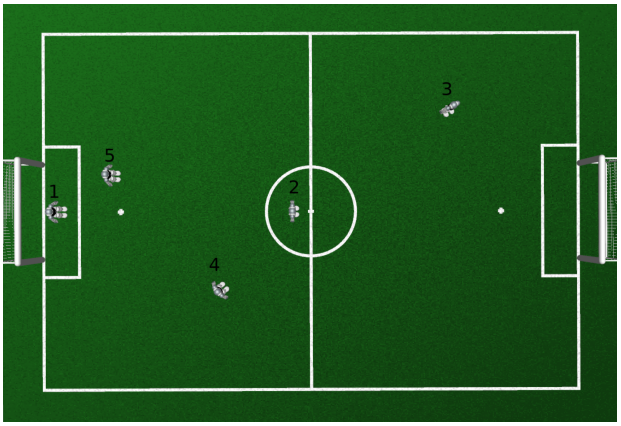

Figure 11. Whistle challenge field example

position estimate for it and determine whether or not it was blown from inside the field or outside of it.

Each whistle attempt (3 points) was scored based on three different criteria as follows:

- Field location (1 point)
- Angular precision (1 point)
- Distance precision (1 point)

Field location depends on whether the robots correctly de-

termined if the whistle was blown inside or outside the field. Angular precision was awarded based on how large the angle between the actual position and the estimated position from the robot was. If it was below five degrees then the full point was awarded. Otherwise the amount awarded was based on a linear scale from 5 to 30 degrees, where anything past 30 degrees was not considered. Distance precision was awarded based on how similar the distance between the robot and its estimated position was in comparison to the robot and the actual whistle position. If the difference was within $\pm 5\%$, then the full point was awarded. Otherwise the amount awarded was based on a linear scale from 5 to 30% deviation, where anything past 30% was not considered. For example, a deviation of 15% would result in a score of 0.6.

The scores from each whistle attempt were added together to create a final score out of 24. The team with the highest score, or the least number of robots in a tie scenario, were ranked as the winner.

8.1. Hardware Selection

The only permitted hardware were the Nao humanoid robots from Softbank. No custom modifications could be made to the hardware. This meant that there were exactly six permissible robots models that could be selected for this challenge, ranging from the initial introductory V1 Nao all the way to the current V6 model.

The robotic hardware, such the gears and joints, remained largely the same across these models. The majority of changes between versions were focused on internal components including the CPU, storage, camera and microphone. The main hardware required for this challenge, since the robots were required to remain stationary, was the CPU for audio processing and the microphone quality. A faster CPU meant more accurate algorithms whilst better microphone quality allowed differing sounds to be separated more easily.

The microphone quality has remained basically the same since the V4 iteration, whilst the CPU performance has only increased. Logically this would mean that the latest version, the V6, would be the optimal choice. And this would be true, assuming Softbank had not decided to radically change the microphone placement on their robots from and including the V5 iteration.

Each NAO robot has a total of four microphones that are all positioned on its head. In iterations before and including the fourth, these microphones were positioned in such a way as to increase the distance between each one and better respond to sounds surrounding the robot in any direction.

This optimised sound localisation operations since it was easier to distinguish when each individual microphone received the same sound. Unfortunately, the same could not be said for sound recognition since most target sound emanates from the front of the robot through people trying to interface with it. This meant that only one microphone received the sound clearly whilst the others did not. As recognition is typically used more than localisation, the microphone positioning was changed to prioritise this for models past the V4.

In terms of a visual comparison, Figure 12 details the current microphone positions for the V5 and later models whilst Figure 13 details the microphone positions for the V4 and prior models.

Figure 12. V5 and later Nao robot microphone positions

Figure 13. V4 and prior Nao robot microphone positions

We hypothesise that utilising the V4 models for sound localisation would have been more accurate, assuming that any unofficial changes made in microphone quality between the upgrades were not significant, as information regarding their changes was not directly provided by the manufacturer. Unfortunately, due to the majority of our V4 models being quite old and inoperable, the team opted to use a set of V5 models instead, promoting consistency amongst whistle measurements.

8.2. Solution

The problem was split into two separate stages comprising sound detection and positioning. Both problems were independent such that as long as the output flowing from detection to positioning remained the same, they could be developed simultaneously.

8.2.1. Detection

This involved first recognising the whistle and then determining its angle and elevation relative to the robot in question. The recognition process had mostly been completed through the regular whistle detection algorithm already used during the soccer games. Two slight modifications were required to allow it to be used more readily however.

The first was allowing it to be operated synchronously so that it may be called and return only when a whistle is heard. Typically it would run asynchronously since whistle occurrences during a soccer game are basically random, but in the whistle challenge they are entirely predictable.

The second modification was to the length of the recorded whistle segment. The current algorithm outputs the entire whistle with around half a second of padding towards the front such that the whistle itself started late and ended early. Since the timing of the whistle was more important that the actual sound itself, this initial padding was removed. The time when the whistle began was then returned.

ALSoundLocalisation Controllability Softbank includes an ALSoundLocalisation module as part of their API for controlling the NAO robots. This module can theoretically detect a sound based on a predefined sensitivity value (0.0 1.0) and return the estimated angle, elevation and confidence relative to the robot that heard it. It is worth noting that this estimate was far from accurate. The specifications state that the accuracy should have a theoretical maximum of 10 degrees provided that the sound is heard within 120 degrees from the front of the robot. Even when tested in an enclosed, controlled environment, the results were erratic at best with the elevation value being essentially useless. There were times when the whistle was blown from the front and the robot declared with nearly complete certainty that it was heard from the back. Ideally if time allowed, the use of this module would be avoided and a custom solution, perhaps treating each robot as an individual microphone itself, would have been used.

The ALSoundLocalisation module can only function asynchronously such that the user must declare a callback function that the module itself will call every time it thinks it has perceived a sound. This means that there is no way to determinatively prove that the sound returned by the callback mechanism was actually the whistle. This was a great annoyance during development. The team developed two alternative methods of substituting this determinative functionality.

The first method involved hearing and recording the entire length of the whistle across all four microphones. Once recorded, the microphones on the NAO robot was disabled and the audio was passed directly to the audio module in the form of a WAV file (no other method was available). The idea behind this was that the ALSoundLocalisation module could perceive the whistle sound as it was passed to the audio module such that it would hear only the whistle exclusively without any background interference. Since the localisation module only heard the whistle, it could be assumed that the localisation callback would only work based on said whistle. Unfortunately this did not work as expected. The ALSoundLocalisation module only worked when the audio file was passed to the audio module, which lent itself to the belief that the localisation process was based on the whistle. However, the actual localisation results were not at all accurate nor representative of anything in general. This was further proven by recording audio when the microphones of the NAO robot were disabled. Instead of getting some error or hearing essentially nothing, only white noise was present. It was not silent and had clear, albeit slight peaks and troughs in its transmission. These are the sounds that were being localised. There was no proof, either technically or through the results, that it was actually hearing the whistle at all.

The second method of determining what localised sound was the whistle was based on the timestamp returned from the whistle. Since the time was relative to the internal clock of the robot, the time returned from our custom whistle detection algorithm could be compared with the time returned from the ALSoundLocalisation module. The sound with the highest confidence that existed within a small time interval epsilon at either side of the whistle timestamp was selected. This is obviously not determinative as it is entirely possible that the localisation module may lag behind and miss this interval, but it was unfortunately the most efficient method available at the time and was selected as a result.

In conclusion, the custom whistle detection algorithm was used to discern the time when the whistle was heard which was used to find the closest corresponding sound returned by the ALSoundLocalisation module. The angle, elevation and associated confidence of such a sound was returned and passed to the second stage of the solution, which was actually positioning the sound.

8.2.2. Positioning

This involved collecting the sound detection results from every available robot and calculating the overall position of the whistle based on them. Since the dimensions of

the field and the position of each robot was known, this could be done entirely mathematically without needing any vision elements. Two plausible and tested solutions were proposed to achieve this, both of which have the same initial processing but differ in the techniques used to generate their final result.

They begin by first creating a unit vector for each robot that points directly in front of them at a default angle of 0 degrees. This vector is then rotated based on the angle between the robot and the whistle. Figure 14 illustrates an example of the processing performed up until this point.

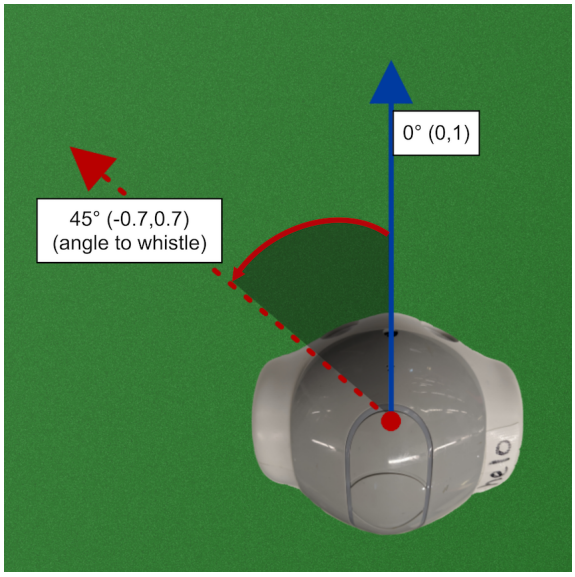

Figure 14. Unit vector rotation process

The resultant rotated unit vector (red) now points directly to the whistle, assuming that the provided angle is correct. This vector will be used throughout the following solution explanations.

Vector Intersections The angle of the robot towards the whistle changes depending on the location of the robot. Therefore if at least two robots exist, the change in this angle relative to their location can be used to locate the whistle. The more robots present, the more times this can be done, resulting in a higher overall accuracy.

For each unique pair of robots, an intersection point is calculated utilising their corresponding rotated unit vectors. This is achieved by implicitly creating another vector for each robot in the pair. This new vector begins at the location of its associated robot and continues infinitely in the direction defined by the rotated unit vector. The point at which these new vectors intersect is the estimated location of the whistle. Figure 15 illustrates this approach.

The resulting collection of intersection points are then

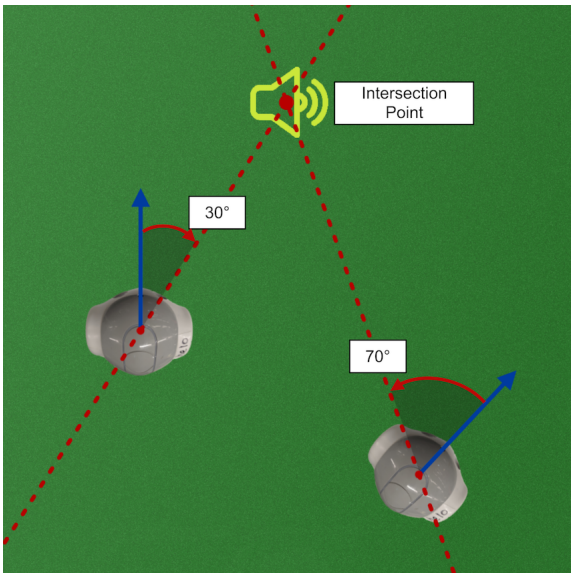

Figure 15. Vector intersection approach

parsed for any obvious outliers before being averaged to result in a final location estimate. Theoretically, this would operate perfectly provided that the initial angles generated by the localisation stage were perfect. Unfortunately this was not the case and the inaccuracies present in the angles were reflected in the intersections. This was alleviated to a degree by including more robots under the assumption that their average would mask the errors to an extent. This method was not selected for use in the demonstration because of this issue, as the fluctuations were too unpredictable to control.

8.2.3. Heat Map Selection

It is clear that the angle generated by the localisation stage has no guarantee of being entirely accurate. This means that the rotated unit vector likely does not point at the sound source, but instead points at another target, hopefully within its general vicinity. The entire placement field was split into a grid to take advantage of this with each cell being given a particular value as per a traditional heat map.

For each rotated unit vector, an implicit whistle vector was generated that shared the same location as its corresponding robot. This vector extends infinitely in the same direction as the unit vector, passing through a multitude of grid cells in the process. If it passed through a cell directly, then that cell would be appended a value of 1. All other cells would be appended a value lower than this by a factor based on the angle between this whistle vector and another that passes through the specific secondary cell in question, both being relative to the robot. If the angle is large, meaning the vector that directly passes through an outside cell has a

direction quite different to that of the whistle vector, then the appended value will be close to zero.

After each rotated unit vector has been parsed, the remaining grid will contain values corresponding to the approximate vicinity of the whistle. Cells with higher values mean that the whistle location is likely to be close. Figure 16 provides an example that illustrates this process.

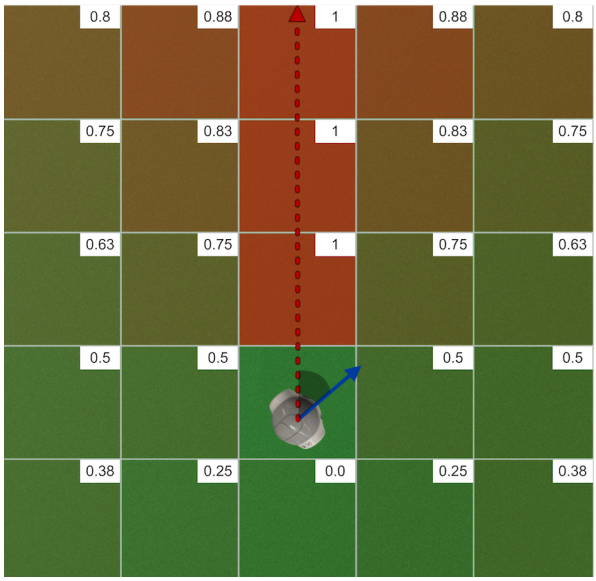

Figure 16. Vector heat map population process

The cell containing the largest value is converted to cartesian coordinates and selected as the approximate location of the whistle. Since the entire cell is being reduced to its centre location, it stands to reason that the smaller the cells are, and therefore the more of them that exist, the more accurate the overall algorithm will become. This is obviously a trade-off with performance since having too many cells will take the algorithm longer to compute, especially considering the intended CPU. There are optimisations that can be implemented to alleviate this, but it cannot be entirely eliminated. In the final version of the algorithm, cells of size 40 by 40 were used to cover the entire area of the soccer field, which was 7400 by 10400, resulting in 48,100 cells in total.

Unfortunately this performance trade-off meant that determining if the whistle location was off-field could become quite demanding if the actual location was not in the immediate outer perimeter. For example, using the figure above, if the whistle was blown outside of this field, such as four sequential blocks away, then at least four extra layers would need to be added to detect it, resulting in an extra 48 cells. To alleviate this issue, a new grid was created that covered an area of 29600 by 41600 with cells of size 250 by 250, resulting in 19,701 cells in total. This larger perimeter grid was run first, and if the whistle was deemed to be within the soccer field, then the smaller, more accurate grid was used afterwards.

This method was selected for use in the demonstration since its approximation approach was more resistant to errors in the localisation stage in comparison to intersection averaging.

8.3. Results

Despite the testing performed prior, this system did not work as intended during the actual demonstration. According to the log files, the processing worked as intended and a result was produced, however there was an issue with its transmission to the central judging computer. We hypothesise that this is likely the fault of our execution approach since our system required an ssh connection initially to be activated instead of being triggered and ran on the robots directly. Losing the ssh connection when the competition began (as all personal systems must be closed to avoid illegal interference) may have been the trigger. As a result, no score was achieved.

9. Concluding Discussion

rUNSWift attained 3rd place in the 2019 robocup SPL. We won 6 of 7 games, losing only to the 2019 champion team, B-Human. We scored a total of 36 goals, while allowing our opponents to score only 5 goals against us. This is a significant improvement over our already strong performance in 2018. We hope to continue this trend and achieve still better performance in 2020.

In 2019 we significantly improved our state estimation and behaviours, while making smaller improvements in many other areas. In 2020 we hope to continue improving our vision system. While this system has seen yearly improvement it still lags behind the level of performance achieved by some other top teams. We aim to automate more of our calibrations, as these act as a considerable distraction during the critical competition period. Finally, we hope to improve on our behaviour system, particularly in the form of passing and positioning.

Acknowledgements

The 2019 team wish to acknowledge the legacy left by previous rUNSWift teams and thank the School of Computer Science and Engineering, University of New South Wales for their continual financial, administrative and financial support to our team. We would like to thank the Nao Devils team for the use of their camera driver, with some modifications. We also wish to pay tribute to other SPL teams that inspired our innovations in the spirit of friendly competition.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bai, G.; Brady, S.; Brameld, K.; Chamela, A.; Collette, J.; Collis-Bird, S.; Hall, B.; Hendriks, K.; Hengst, B.; Jones, E.; Pagnucco, M.; Sammut, C.; Schmidt, P.; Smith, H.; Wiley, T.; Wondo, A.; and Wong, V. 2017. runswift 2017 team report and code release. Technical report, The University of New South Wales.

Bradley, D., and Roth, G. 2007. Adaptive thresholding using the integral image. *Journal of graphics tools* 12(2):13–21.

Brameld, K.; Hamersley, F.; Jones, E.; Kaur, T.; Li, L.; Lu, W.; Pagnucco, M.; Sammut, C.; Sheh, Q.; Schmidt, P.; Wiley, T.; Wondo, A.; and Yang, K. 2018. runswift 2018 team report and code release. Technical report, The University of New South Wales.

Hengst, B. 2014. rUNSWift Walk2014 report. : http://cgi.cse.unsw.edu.au/'tilde'robocup/2014championteampaperreports/20140930-bernhard.hengst-walk2014report.pdf, University of New South Wales.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Nielsen, F. 2012. k-mle: A fast algorithm for learning statistical mixture models. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, 869–872. IEEE.

Rosenfeld, A., and Pfaltz, J. L. 1966. Sequential operations in digital picture processing. *Journal of the ACM (JACM)* 13(4):471–494.