Visual Compass for Mobile Robot Orientation Estimation

Patrick Baumann (7047847), Jan Gutsche (7061491),
Benedikt Ostendorf (6981708), and Florian Vahl (7031630)

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics, Group TAMS
Vogt-Kölln-Straße 30, D-22527 Hamburg
{7baumann,7gutsche,7ostendo,7vahl}@informatik.uni-hamburg.de

Abstract. In the RoboCup Humanoid Soccer League, the orientation of the autonomous mobile robot on the field plane is critical i.e. to distinguish between the own and the opponent's goal. We propose our *Visual Compass* approach using only the camera image to estimate the robot's orientation on a symmetrical soccer field. Our modular Open Source implementation is written in Python and supports the ROS framework. The approach is discussed and evaluated against a comprehensive data set recorded on a tournament RoboCup soccer field.

Keywords: RoboCup · Robot Localization · Computer Vision

1 Introduction

written by Jan Gutsche

Humanoid robotics is an ongoing field of research [9,26,13]. For promoting and motivating research and development of mobile robotics and AI, the idea of the RoboCup competition and humanoid robots playing soccer has been around since the early 1990s [16]. During a game of soccer, it is critical for the autonomous mobile robot to know its own orientation on the field plane, i.e. to distinguish between the own and the opponent's goal.

The set of rules of the humanoid soccer league will be adapted each year to match the FIFA rules by 2050 [17] and restricts the usage to only human-like senses [18]. Obtaining the orientation of the robot on the field plane can be part of the self-localization system and is not trivial to determine since the field itself is symmetric (color-coded landmark poles were removed in the set of rules in 2013 [19, Section 1.3]). Relative orientation estimations of the robot's odometry or IMU (Inertial Measurement Unit) can drift over time because of accumulating measurement errors. This especially becomes a problem after the robot has fallen (see Section 2).

Therefore a robust solution for obtaining the orientation, which is independent of the game's progress or the robot's state, is needed. This paper promotes our modular *Visual Compass* approach. It utilizes well-known feature recognition algorithms to create a pre-recorded map of the environment the robot can use to find its own orientation during the game (see Section 3). As those algorithms are robust against rotation, scaling and position inside the image, it is possible to recognize those features from different positions on the field (see Section 4).

The Visual Compass approach presented in this paper has been developed in the scope of the course *"Praktikum RoboCup – Fußballspielende Roboter"* in strong relation to the *Hamburg Bit-Bots RoboCup Humanoid Soccer league kid-size* team [8].

Our approach is implemented as ROS nodes in Python using OpenCV among other libraries. ROS (Robot Operating System) is a middleware originally developed by Willow Garage for robotic systems [15]. The ROS framework provides message passing between nodes as a communication layer upon the operating system used on the robot. Therefore ROS nodes are highly interchangeable and the integration of nodes into the own software stack becomes straightforward. Besides that, ROS also provides debugging and visualization tools. Our ROS nodes are using standardized ROS-messages proposed by Bestmann et. al. in 2017 for the Humanoid Soccer League [4].

To introduce the topic, we give an overview of various approaches on the calculation of the orientation (see Section 2.1). After that, we shortly describe three different algorithms for feature recognition: AKAZE, ORB and SIFT (see Section 2.2). In Section 3, we describe our architecture in detail and illustrate the angle and confidence calculation. Afterwards, we evaluate our results in Section 4. Finally, Section 5 gives a short summary and an outlook on future work on this topic.

2 Related Work

written by Jan Gutsche

In this Section, first, we discuss related approaches for solving the similar problem of estimating the robot's orientation. Second, we give a short overview of the used image feature detection algorithms.

2.1 Related Approaches

Several approaches on orientation estimation are also based on feature detection of camera images.

Visual odometry as proposed in [14] is a robust SLAM (Synchronous Localization and Mapping) mechanism for autonomous vehicles and robots such as Mars rovers [12]. Visual odometry analyses the displacement of detected and matched image features between image frame pairs, called optical flow. Thus an estimation of trajectories of the camera and further the position and orientation of the camera can be achieved. Nistér et. al. demonstrated that their approach is suitable for estimating the position of the vehicle or robot for hundreds of meters, "based only on visual input from relatively small field of view cameras" [14, Section 6]. Since visual odometry only uses the camera image as an input, it is independent of other sensor information such as the GPS signal (Global Positioning System) or IMU data. This allows position estimation even in environments with poor GPS signal quality (e.g. indoor). An improvement of the position and orientation estimation can be achieved by fusing all these input signals

The application of this approach in humanoid robotics is presumably problematic, because it is very hard to exactly track the camera movement while the robot falls.

In this scenario with fast motion, the camera captures probably only grass of the field with few to zero recognizable features. The feature comparison of such two images could be inaccurate. Since visual odometry relies on the correct detection over the whole image sequence, the measurement error would continue on and could accumulate with further inaccuracies. Therefore the estimation could be shifted and this result could lead to false decisions.

Ceiling detection. The humanoid league team *WF-Wolves* developed in principle a very similar approach to ours [28]. They used SIFT as an image feature detection algorithm to match those features with prerecorded images of the ceiling with known orientation. Therefore this orientation estimation does not have the disadvantages of the relative measurements, e.g. visual odometry or IMU. Unfortunately, their approach was never evaluated since. Also, this method is probably not reliable in case of a symmetric or homogeneous ceiling. Further, obtaining the robot's orientation by this approach requires the robot to look straight upwards to the ceiling, which does happen rarely during a game. Thus a special behavior for the camera motors is needed.

Color histogram. Another approach based on the camera image, but does not share the drawbacks of relative methods, is using color histograms. Color histograms simply show the frequency of every color of the color space (e.g. RGB, HSV) in the image. Therefore histograms of images recorded during the soccer game could be matched against prerecorded ones. Those prerecorded histograms are linked to the orientation, so the robot's orientation during the game can be deduced. Since color histograms do not model any structural information about the scene, two completely unrelated images could have very similar histograms. For this approach we were also not able to reference any evaluation data.

Odometry estimates position changes over time by using information from the movement actuators of the robot. For example in wheeled robots, the measured rotation of wheel encoders can be used to determine the position relative to the starting point. Thought noisy, Thompson et. al. showed that odometry is also applicable for bipedal humanoid robots [25].

In case of a fallen robot, this method has a related problem as visual odometry. Odometry only uses the measured movements of the robot's joints for position and orientation estimations, assuming that the robot always stands upright. So after a fall, the robots position and orientation could have changed by comparison to the originally tracked ones. Since odometry is based on a sequence of measurements, possible errors accumulate and lead to false estimations.

IMU (inertial measurement unit) contains accelerometers and gyroscopes for sensing relative position, velocity, acceleration and rotational rate of the observed object in six-degrees-of-freedom [21, 4.1.7 p.121]. The unit often also includes a magnetic compass, but this sensor is not allowed in the humanoid soccer league.

The position and orientation estimations of an IMU are also based on a relative sequence of measurements and therefore prune to accumulating errors. These errors could appear while the fast movements of a robot's fall.

Several other approaches (e.g. Fourier transformation or convolutional autoencoder) were considered as possible solutions for the problem. But we have chosen our Visual Compass approach as described further in Section 3, because we wanted to avoid any accumulating errors and liked the idea of using image feature detection.

2.2 Feature Detection

written by Florian Vahl

In the following part, we are looking at the local image feature detection algorithms SIFT, ORB and AKAZE. All described algorithms are mostly invariant against translation, rotation, scale and light changes. The detected image features are corners in the output of an edge detection algorithm [5].

SIFT (scale-invariant feature transform) uses a *difference of Gaussians* band-pass filter to detect the local image features. Features are described by local histograms in multiple directions around the feature as shown in [11]. Each feature is represented by a description vector. Descriptions vectors can be compared using their euclidean distance.

AKAZE (accelerated KAZE) is an improved version of the KAZE algorithm, that is similar to SIFT. Instead of Gaussian blurring, it uses means of nonlinear diffusion filtering, which respects certain image edges in the blurring process [1].

ORB (oriented FAST and rotated BRIEF) is a combination of the FAST [20] called corner detection algorithm and a probabilistic rBRIEF descriptor. Each image feature found by FAST gets described by a binary description vector generated by rBRIEF [2]. Similar features are compared by comparing their hemming distance.

3 Approach

written by Florian Vahl

To solve the challenges described in Section 1, we are using image features detected by the SIFT, ORB or AKAZE algorithms. Our architecture is generally independent of the used feature detection algorithm.

A map which is linking the image features with an orientation describes the robot's environment beyond the field boundary. In the game, image features found in the current image are matched with the features saved in the map. This is used to generate an orientation estimation using the orientation of the matched features.

3.1 Training

Before the game starts, a map gets generated. The described principle is shown in the *Training* segment of Figure 1.

The robot is placed with a predefined orientation on one or more locations on the field. Then the robot records images in every direction, except directions without any benefit, like straight up or those which only show the field itself. Due to its symmetrical design, features found on the field don't contain enough useful information. Also, we try to search for very distant features to minimize the errors introduced by the translation

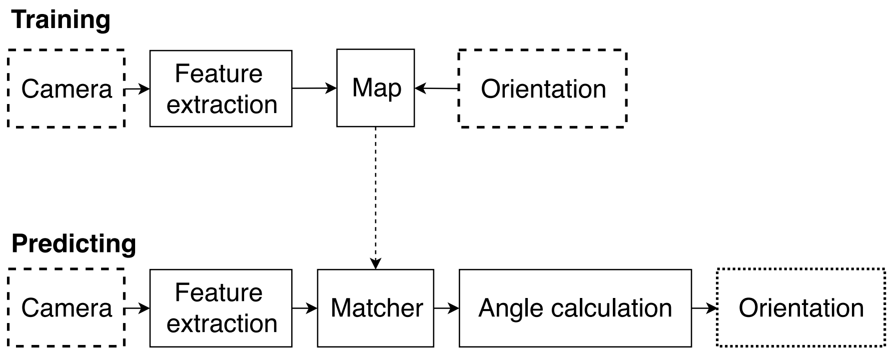

Fig. 1. The Visual Compass pipeline

of the robot on the field. The usage of multiple field locations in the map-generation process should also reduce the effect of the robot's translation on the field.

In the map generation process, the robot's orientation on the field is known. This allows us to annotate each detected image feature with an angle. The angle gets calculated by combining the known orientation of the robot with the transform from the robot's base footprint frame (as described in REP 120 [24]) to the camera frame and the position of the feature in the image. The transform is done using ROS and tf2 forward kinematics [7]. Due to a camera calibration matrix, we are also able to calculate the angle offset regarding the feature position in the image. This is important if the robot uses a lens with a large field of view.

Each detected image feature is described by a description vector generated by the SIFT, ORB or AKAZE algorithm. The pickle encoded file contains the description vectors of the detected features and a list of corresponding annotation angles. This feature map file can be spread between the robots before the game starts, to minimize the map generation effort.

3.2 Prediction

In the game, the Visual Compass runs as shown in the *Prediction* segment of Figure 1. It uses the selected feature detection algorithm to detect all significant image features in the current camera image.

Matching The descriptors of these features get matched with corresponding features in the feature map via a k-nearest neighbor algorithm. Instead of comparing the descriptor of the found image feature only to the nearest neighbor and use global distance threshold, the distances of the nearest and the second nearest neighbor are compared with each other like D. Lowe et. al. described in [11]. This makes a global threshold obsolete. The description of a feature is rotation and scale-invariant. This is important to our implementation because we want to identify as many features as possible, even from different perspectives, due to the translational movement of the robot on the field.

Orientation estimation (written by Benedikt Ostendorf)

After the matching stage, we can associate the features found in the current camera image with the angles from our feature map, which contains known good values after initialization. At this stage there usually are many such features with different associations. These need to be distilled down to a single direction together with a confidence value to signal the quality of the prediction.

The algorithm used to reduce the results of the matching stage into one final result is quite simple. We interpret the directions as two-dimensional vectors with unit length. Any two such vectors can be added up, resulting in a new vector pointing in the average direction of both, for some definition of average. Because vector addition is commutative and associative we can just add up all the directions as vectors. The resulting vector can easily be converted back into a direction. This direction is used as the output of our Visual Compass.

Together with the predicted direction we also want to compute a confidence measure for the prediction. Such a measure should increase when measurements agree and decrease when measurements disagree. In other words, the change in the measure should be proportional to the strength of agreement in the measurements. It is easy to show that the magnitude of the vector described above is such a measure.

Suppose we convert directions into vectors of unit length as described above. The magnitude of the sum of those vectors is maximal if the directions are the same. Similarly, the magnitude of the sum of both vectors is minimal if the directions are opposites of another. Additionally, the magnitude of the sum of two vectors increases as the difference in direction, approximated by the dot product, decreases. Therefore the magnitude of such a vector can be used as a confidence measure. The confidence measure should be normalized to the interval [0,1] for ease of use. This is achieved by dividing the magnitude of our result vector by the maximal magnitude that can be reached. This unlikely scenario happens when all features found in the camera image are associated with the same direction.

To protect the compass from bad predictions due to a few number of features we scale down the confidence based on the number of features. This is done by multiplying the confidence value described above with a value that varies with the number of features. For this value we use

$$1 - e^{(\frac{-n}{s})} \tag{1}$$

where n is the number of features used in the calculation and s is some scalar from the settings of our compass. This value is close to 0 for small values of n and close to 1 for large values of n. By choosing an appropriate value for s it is possible to require any number of features for a prediction with high confidence. In our implementation s is based on the number of features that were detected in the training stage. Should we encounter significantly fewer features during the prediction stage the confidence will be small.

The result of our Visual Compass prediction is a pair containing a direction from 0 to 2π and a confidence from 0 to 1. These values are acquired by summing up all directions from the matching stage as two-dimensional vectors. The direction of this resulting vector is used as the reported direction, together with the normalized magnitude as a confidence value.

3.3 Architecture

written by Florian Vahl

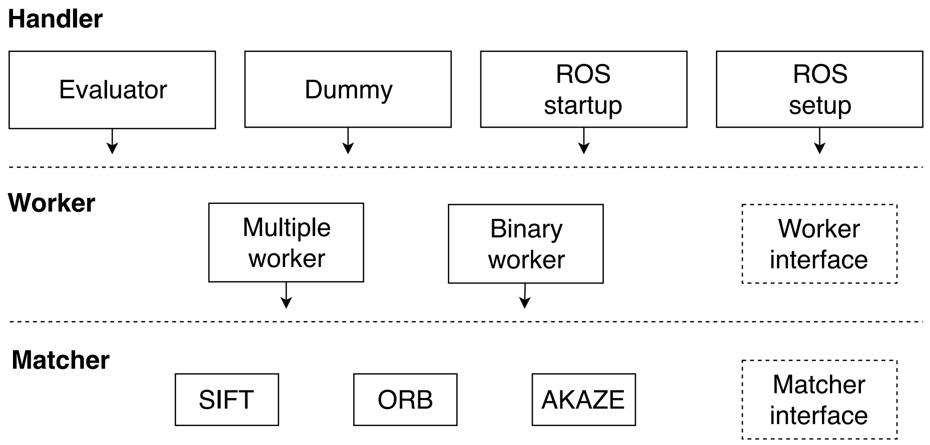

Fig. 2. The different layers of the Visual Compass software architecture. The handler layer consists of handlers to run the Visual Compass in different environments. In the worker layer, different implementations of the Visual Compass are integrated. The matcher layer abstracts the different feature detection/matching algorithms.

Our approach is implemented in Python, supporting the ROS framework. For image processing, we are using the libraries OpenCV [6] and NumPy [27]. OpenCV is also used for the ORB and AKAZE feature detection algorithm, while SIFT uses the Silx [22] library. All implementations of the feature detection algorithms are based on OpenCL [23], which allows multi-core CPU and also GPU execution, to improve the processing speed.

The architecture of the Visual Compass consists of multiple abstraction layers as seen in Figure 2. These layers allow the compass to operate in different software environments like ROS or our evaluator. On the other side, this modular approach allows us to test and compare different approaches, like the multiple/binary worker or the different feature detection algorithms.

Handler As seen in Figure 2 the Visual Compass supports many different environments. Each of these environments has its own handler that communicates with the worker instance.

First of all, there is a ROS independent *Dummy handler*. It is used to simply run a Visual Compass instance without the need of a whole ROS environment. The *Dummy handler* supports a webcam or image set as the image source. The map creation is simply managed via a CLI control.

The Evaluator is also ROS independent. It executes the Visual Compass instance on a predefined test data set. See evaluation in Section 4 for more details.

The ROS interface for the Visual Compass consists of two separate ROS nodes. The *Visual Compass setup* node coordinates the map creation in the robot environment. First of all, it manages the user input that is necessary for the map creation such as the robot's current position/orientation. It also sets the robot's head behavior node in a specific record mode and receives a ROS message from it, if the head-mounted camera reaches a so far uncovered perspective. Beside the image topic, it also subscribes to the `camera_info` topic to get the camera calibration. This whole information is passed into the Visual Compass implementation. If the recording is finished, the feature map is dumped in a pickle file and the user gets notified.

The *Visual Compass startup* node runs the Visual Compass during the game. It subscribes to the ROS topics for the camera image and calibration. In addition, it loads the feature map from a pickle file and passes everything to the Visual Compass implementation.

Worker The layer underneath the handler layer is called the worker layer as seen in Figure 2. This is where the different implementations of the Visual Compass are located. Each of these implementations implements the worker interface. The worker interface describes methods to get/set/append the feature map, to run the compass on a specific image or to set a configuration dictionary.

The most used implementation is the *Multiple worker*. It calculates an orientation of the robot on the field plane in radians, in contrast to the *Binary worker* that only determines which field side the robot is facing. So the *Binary worker* is only able to output the values $\{0,\pi\}$ with a confidence value. Due to improvements in the development process of the *Multiple worker*, the functionality of the *Binary worker* became a subset of the *Multiple worker*. Consequently, we focused on improving the *Multiple worker* and the *Binary worker* got deprecated.

Matcher The matcher layer in Figure 2 consist of classes for the different feature detection algorithms. Every class implements the matcher interface. The matcher interface defines functions for the image feature detection, the matching of feature descriptors or feature algorithm-specific debug image generation. Matcher implementations exist for the SIFT, ORB and AKAZE algorithms.

4 Evaluation

written by Patrick Baumann

This Section aims to quantify the quality of the Visual Compass. First, we will give a detailed look into the precision of the compass. In Section 4.4, we will focus on the runtime performance of the different variances of the compass. As stated in Section 3, the Visual Compass uses images taken *before* a game in a training phase to create a map of its environment. It then uses those features to recognize features from this map in the images taken *during* the game. For precision evaluation purposes, we decided to take unstitched panoramic images from the playing field of the RoboCup German Open in May 2019 in Magdeburg. The images were taken on a grid with 1 m density from all over the field, resulting in 70 sets of images (the field has a size of $6\,\mathrm{m}\times 9\,\mathrm{m}$) with 16 images per location.

For the evaluation, a central set of 16 images is taken as the source for constructing the feature map. Then, the Visual Compass is run against all the images taken from different positions all over the field. The exact viewing angles of the test images are known, thus, it is possible to quantify the error the compass makes on processing each image.

The compass has two output values: the angle in which the robot is supposedly looking and the confidence value to quantify the trust in the calculated angle (Section 3.2). During the evaluation process, we specified threshold values for both parameters, resulting in a 2×2-Matrix of outcomes for each picture in the evaluation process. This *confusion matrix* (see Figure 4) contains

– true positives: accurate angle values with high confidence ratings,
– true negatives: inaccurate angles with low confidence ratings,
– false negatives: accurate angles with low confidence ratings and
– false positives: inaccurate angles with high confidence ratings.

In the following sections, we take a look at the general precision performance, on some specific issues that emerged during the evaluation process and discuss its shortcomings. Finally, we evaluate the runtime performance of the compass.

4.1 General precision

The ability of the Visual Compass to perform well is not depending on the parameter set that is used. This general performance can be visualized in a Receiver Operating Characteristic (ROC). The ROC's idea is that "a classification model can be tuned by setting an appropriate threshold value to operate at a desired [... false positive rate]" [10, p. 130]. The more a ROC curve fits in the top left corner, thus having many true positives while keeping the false positives low, the better is the overall performance of that receiver.

Comparing the three characteristics in Figure 3 shows the strengths of the different algorithms: AKAZE keeps the false positive rate at 0 for the longest, but struggles to get the true positive rate up, even when tolerating some false positives. SIFT and ORB for the most part develop in parallel. Since SIFT maximizes the area under the curve, it has the best overall performance of the three.

During a game situation, the false positives – being sure to look in one direction while actually looking in another direction – would trigger a disadvantageous action and therefore come at the highest cost. A low confidence rating would just result in a need for evaluating pictures by looking in other directions with higher confidence. Because of that, we focused on keeping the false positive rate low when deciding for a parameter set to work with. Figure 4 shows the confidence matrix for a confidence threshold of 0.5.

From all 1120 images evaluated from the different positions on the field, the SIFT algorithm manages to retrieve the correct angle of 639 images within a 90° window (45° left and right of the correct angle of the image) with high confidence. With this configuration, it is possible to keep the false positives at only 9 images (0.8%). Regardless of the confidence, about 18% of the analyzed images resulted in a wrong angle output. This shows, that (1) the Visual Compass has a good ability to find the right direction and (2) that the confidence value, filtering out the wrong values, actually works as intended.

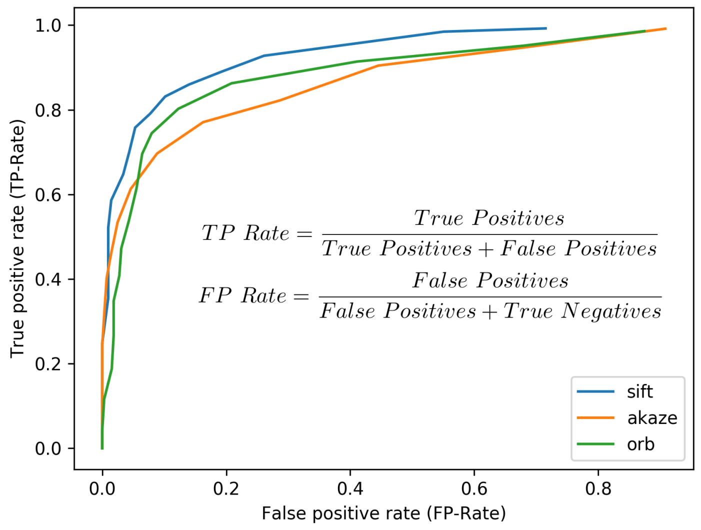

Fig. 3. Receiver Operating Characteristic (ROC) of the three algorithms *SIFT*, *AKAZE* and *ORB*. Since *SIFT* maximizes the area under the curve, it has the best trade-off performance when maximizing the true positives while minimizing the false positives.

Like the ROC curve, the confusion matrix shows that the SIFT algorithm has the best overall performance of the three. While ORB appears to have bigger problems with retrieving the correct angles, it is different with AKAZE: The high count of false negatives point to a low confidence value in general.

4.2 Detailed view: Strengths and weaknesses of the Visual Compass

Until now, the analysis focused on the overall precision of the compass. In this Section, special issues of the Visual Compass will be addressed which emerged during the evaluation process. We focus on the SIFT algorithm, as it obtains the best results.

In the general precision evaluation, we found 9 false positives, three of which are shown in Figure 4 (the circled arrows). The correct direction is north-west, the three false positives, however, are pointing north, north-east. All can be found in the north-eastern corner of the field. Presumably, a perspective distortion is the reason for the errors. The feature map was created by pictures taken from the center of the field. From a central position, the recognized perimeter advertising board is found in a north, north-eastern direction, so this angle gets associated with it in the feature map. When the algorithm recognizes the ads during the evaluation, it outputs that (north, north-east), even though the actual direction in which the picture is pointing is north-west. So actually, the compass' feature assignment is correct, therefore the high confidence makes perfect sense. This behavior can be determined for all 9 false positives of the SIFT algorithm with a confidence threshold of 0.5. This problem would become smaller if the compass used features that are further away from the field.

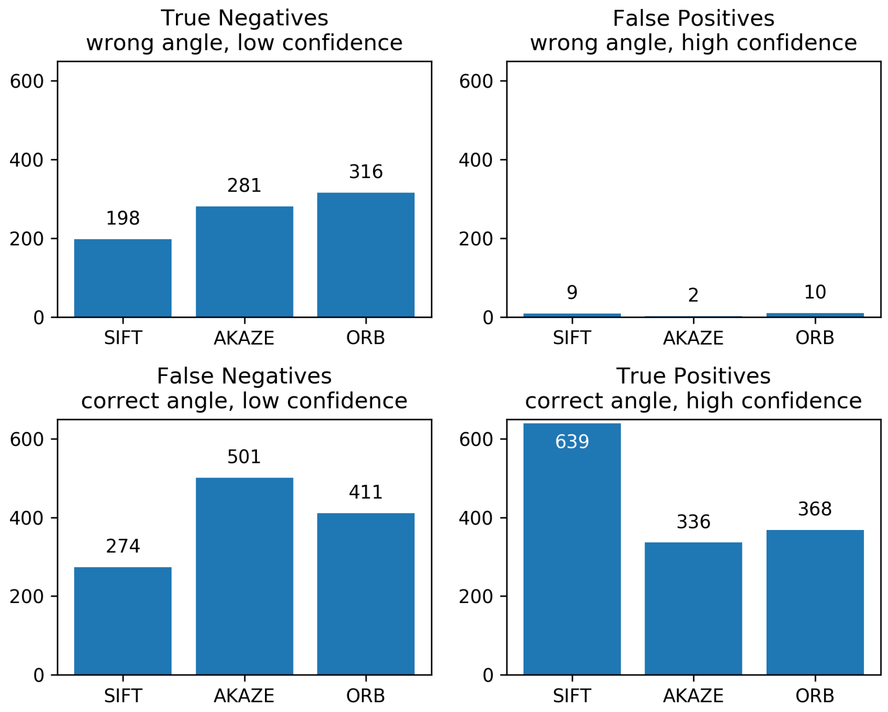

Fig. 4. Confusion matrix of the compass using different algorithms with confidence threshold of 0.5.

Figure 5 hints at another interesting point: The confidences in the left goal area are very low. Manual evaluation of the corresponding images shows that those images were taken looking into the goal net. The feature retrieval algorithm finds many features in the edges and nodes of the net, but is presumably not able to retrieve those in the feature map because of the far distance from the center of the field. Additionally, the same features can be found symmetrically in the right goal, resulting in a low confidence again.

4.3 Restrictions of the precision evaluation and future analyses

Our evaluation process has some restrictions. The data was taken under idealized circumstances. During a game, images are blurred due to the movement of the camera. As the robot is moving, shooting and presumably falling during a game, the images are shot with totally different views, not just from upright. Even though the background changed during the recording process of the evaluation sets, the background in the hall might change more during the game. Since those new features are not recognizable in the feature map, we expect a lower confidence while the directions remain correct – this must be subject to future evaluation. Finally, future research should also investigate further into the low confidences and angles to confirm or reject the hypotheses presented in Section 4.2.

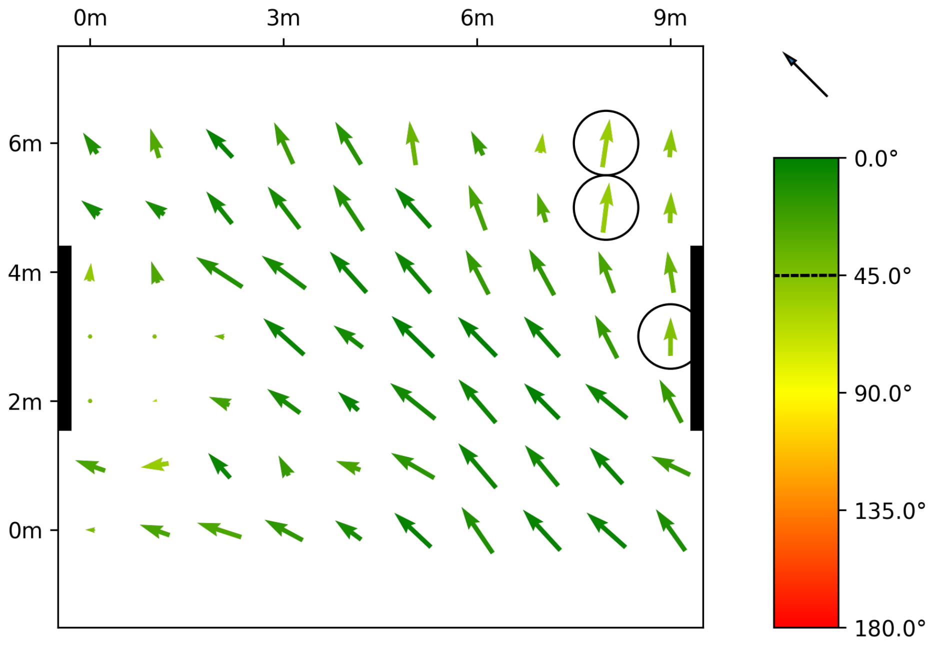

Fig. 5. Evaluation of 70 images: The compass learns its surroundings through one panoramic image from the center position, it then tries to find the direction in which the other images are pointing. Each position on the field is evaluated for itself. The black bars on the left and right symbolize the goals. Arrow directions show the calculated direction, arrow lengths show the confidence. Arrow colors show the correctness of the calculated direction, compared to the correct direction indicated by the arrow above the colorbar. Arrows with circles indicate false positives.

In summary, the precision evaluation shows that the compass is functioning, even though it has its limitations. In most cases, these limitations result in low confidences, not wrong angle associations with high confidences. The Visual Compass can thus be considered stable and working.

4.4 Timing evaluation

written by Florian Vahl

To evaluate the performance of the different feature detection algorithms, represented by the matchers as described in Section 3.3, our setup consisted of an Intel® Core^{TM} i7-4810MQ CPU @ 2.80GHz that runs the Visual Compass in a CPU only mode via the dummy handler on an image set of 1000 images. The dummy handler is used for this experiment, because we want the different approaches to operate without any ROS overhead.

The measured absolute performance isn't comparable to the in-game performance of the Visual Compass. For repeatability reasons, the Visual Compass runs on a clean system without any additional load. Also, the Visual Compass is able to run on many different devices. E.g. in the *Wolfgang* robot from Hamburg Bit-Bots it could be deployed on an Intel NUC, an ODroid XU4 or an NVidia Jetson TX2 as described in the Team Description Paper [3]. These are the main reasons why this timing evaluation focuses mainly on the comparison of the different feature detection algorithms instead of in-game run-time values.

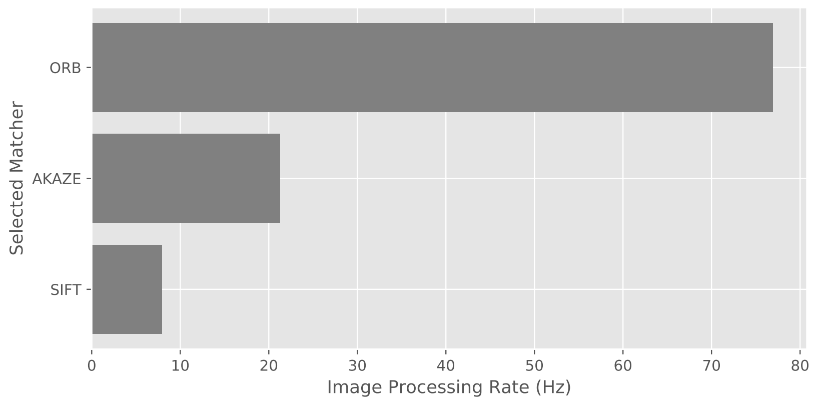

Fig. 6. Processing rates for the ORB, AKAZE and SIFT feature detection algorithms.

As seen in Figure 6 the ORB algorithm reaches a processing rate of 76.9 Hz and performs the best by outperforming the AKAZE algorithm by a factor of 3.6. The probabilistic approach of ORB also performs 9.7 times faster than the SIFT algorithm, which had the worst run-time.

Considering the previous evaluation of the compass precision in Section 4.1, the accuracy benefit of SIFT has to be compared to the run-time benefit of ORB, which only performed slightly worse precision wise. This opens more possibilities for future evaluation.

5 Conclusion and Future Work

written by Benedikt Ostendorf

5.1 Future Work

As seen in the evaluation (Section 4) there is still room for improvement with our Visual Compass. The Visual Compass spends most time extracting features from camera images. Because of that, the runtime of our compass cannot be improved significantly, unless one would choose to improve an existing or use a faster feature extraction algorithm.

What can be improved upon is the quality of our prediction. One such improvement involves the feature to angle mapping that is done in the compass. In order to predict a direction, the compass uses the angle of the features introduced in the initialization stage. If the compass encounters the same feature in the matching stage, its angle can be looked up from our feature map.

But there exists another angle that our implementation ignores. The features that are extracted from the image in the prediction stage can be to the left or right of the image as well. Dependent on the field of view of the robot's camera there could be a discrepancy of up to 40 degrees between the angle we look up in the map and the angle actually present.

Taking this offset into account should improve the quality of our prediction. The improvement in confidence can be calculated for ideal conditions. Assume all angles in the map and all extracted features are correct, and features appear equally likely at all offsets. The algorithm to calculate the confidence converts angles into vectors of unit length and sums them up. In our implementation, mapping these vectors would draw a semicircle dependent on the camera field of view, because all feature offsets are equally likely. With the correction of the feature offset and under ideal conditions, all vectors point in the same direction.

Let f be the maximum offset of a feature. Assuming infinite features and thus using integration instead of summation, the magnitude of the result in our implementation is

$$\int_{-f}^{f} \cos(x)dx = 2 \cdot \sin(f) \tag{2}$$

instead of simply

$$\int_{-f}^{f} 1dx = 2f \tag{3}$$

with angle correction. Converting into a percentage gives

$$\frac{f}{\sin(f)} - 1 \tag{4}$$

Inserting appropriate values tells us that under ideal conditions and with a field of view of 90° we can get a 10% improvement.

Under normal circumstances, it is not the case that features are detected everywhere. In the case of a feature imbalance the improvement would be even better, because the predicted direction would be less sensitive to the location of found features.

Another thing that can lead to problems with correct compass predictions is the potential absence of features. Should something about the field change or if the map isn't initialized correctly, the compass might make a wrong decision based on a few wrong features, or don't return a useful result at all.

To make sure that there are features in the map to match against, the map could be updated in the game. If a prediction for a picture is requested and the compass is confident about its prediction it can add new features from the current input to the map. Such a procedure could make the compass robust against changes that could happen during the game. It also could decrease the accuracy of the compass by repeatedly making wrong predictions after they have been added to the compasses feature map.

Whether or not continuously updating the map is helpful to the performance of the compass should be tested in practice. Even in the case that updating the feature map of the compass introduces more errors, this procedure could be used in parallel to the simple compass implementation. Should the normal compass return a uncertain result the updated feature map could be used to improve the prediction quality.

The above discussed changes conclude the improvements to the quality of the Visual Compass. As mentioned in Section 4 further evaluation of the compass can still be done. Real-time requirements and too few true positives may decrease the prediction's usefulness. Therefore it is still to be determined if the compass can reproduce the results of the evaluation under real-world conditions.

5.2 Conclusion

At the beginning of the semester, we set out to partly solve the self-localization problem for robots using only human-like sensory data. The goal was to produce a Visual Compass that can be used in future matches of the *Hamburg Bit-Bots* Team. Over the course of the semester, we looked at different existing approaches to draw inspiration from. We looked at different feature extraction algorithms and build a proof of concept binary compass. This early version was improved upon as described in the previous sections, resulting in a Visual Compass that can work with multiple reference images and various feature extraction backends.

References

1. Alcantarilla, P.F., Bartoli, A., Davison, A.J.: KAZE features. In: European Conference on Computer Vision. pp. 214–227. Springer (2012)
2. Andersson, O., Reyna Marquez, S.: A comparison of object detection algorithms using unmanipulated testing images: Comparing sift, kaze, akaze and orb (2016)
3. Bestmann, M., Brandt, H., Engelke, T., Fiedler, N., Gabel, A., Güldenstein, J., Hagge, J., Hartfill, J., Lorenz, T., Heuer, T., Poppinga, M., Salamanca, I., Speck, D.: Hamburg Bit-Bots and WF Wolves Team Description for RoboCup 2019 Humanoid TeenSize (2019)
4. Bestmann, M., Hendrich, N., Wasserfall, F.: Ros for humanoid soccer robots. In: The 12th Workshop on Humanoid Soccer Robots at 17th IEEE-RAS International Conference on Humanoid Robots (2017)
5. Canny, J.: A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence (6), 679–698 (1986)
6. Culjak, I., Abram, D., Pribanic, T., Dzapo, H., Cifrek, M.: A brief introduction to OpenCV. In: 2012 proceedings of the 35th international convention MIPRO. pp. 1725–1730. IEEE (2012)
7. Foote, T.: tf: The transform library. In: Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on. pp. 1–6. Open-Source Software workshop (April 2013). https://doi.org/10.1109/TePRA.2013.6556373
8. Hamburg Bit-Bots Team: Hamburg Bit-Bots website, `https://bit-bots.de`, accessed 2019-08-21
9. Kaneko, K., Kaminaga, H., Sakaguchi, T., Kajita, S., Morisawa, M., Kumagai, I., Kanehiro, F.: Humanoid Robot HRP-5P: An Electrically Actuated Humanoid Robot With High-Power and Wide-Range Joints. IEEE Robotics and Automation Letters **4**(2), 1431–1438 (2019)
10. Kantardzic, M.: Data mining: concepts, models, methods, and algorithms. John Wiley & Sons, Hoboken, New Jersey, USA (2011)
11. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2), 91–110 (2004)
12. Maimone, M., Cheng, Y., Matthies, L.: Two years of visual odometry on the mars exploration rovers. Journal of Field Robotics **24**(3), 169–186 (2007)

13. Meghdari, A., Alemi, M., Zakipour, M., Kashanian, S.A.: Design and realization of a sign language educational humanoid robot. Journal of Intelligent & Robotic Systems **95**(1), 3–17 (2019)
14. Nister, D., Naroditsky, O., Bergen, J.: Visual odometry. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 1, pp. I–I (2004). https://doi.org/10.1109/CVPR.2004.1315094
15. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
16. RoboCup Federation: A Brief History of RoboCup, `https://www.robocup.org/a_brief_history_of_robocup`, accessed 2019-08-21
17. RoboCup Humanoid League: The official website of the RoboCup Humanoid League, `https://humanoid.robocup.org/`, accessed 2019-08-21
18. RoboCup Soccer Humanoid League: RoboCup Soccer Humanoid League Laws of the Game 2018/2019, `http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Rules-changesMarked.pdf`, accessed 2019-08-25
19. RoboCup Soccer Humanoid League: RoboCup Soccer Humanoid League Rules and Setup, `http://www.robocuphumanoid.org/wp-content/uploads/HumanoidLeagueRules2013-05-28-with-changes.pdf`, accessed 2019-08-21
20. Rosten, E., Drummond, T.: Machine Learning for High-Speed Corner Detection. In: ECCV (2006)
21. Siegwart, R., Nourbakhsh, I.R., Scaramuzza, D.: Introduction to autonomous mobile robots. MIT press (2011)
22. Solé, V. A., e.a.: silx-kit: Scientific Library for eXperimentalists, `http://www.silx.org/`, accessed 2019-08-29
23. Stone, J.E., Gohara, D., Shi, G.: OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering **12**(3), 66 (2010)
24. Thomas Moulard: Coordinate frames for humanoid robots, `https://www.ros.org/reps/rep-0120.html#base-footprint`, accessed 2019-08-29
25. Thompson, S., Kagami, S.: Humanoid robot localisation using stereo vision. In: 5th IEEE-RAS International Conference on Humanoid Robots, 2005. pp. 19–25. IEEE (2005)
26. Tian, L., Thalmann, N.M., Thalmann, D., Fang, Z., Zheng, J.: Object Grasping of Humanoid Robot Based on YOLO. In: Gavrilova, M., Chang, J., Thalmann, N.M., Hitzer, E., Ishikawa, H. (eds.) Advances in Computer Graphics. pp. 476–482. Springer International Publishing, Cham (2019)
27. Van Der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22 (2011)
28. WF-wolves RoboCup-Team: Wf-wolves robocup team: source code of visual compass, `https://humanoid.wf-wolves.de/HeadSoftware/visual_compass/tree/4c41950ae8465910ca1935c3598e61e680823714`, accessed 2019-08-19

Get the code:
`https://github.com/bit-bots/bitbots_navigation/tree/master/bitbots_visual_compass`