Final Project Report

Edward Banner*

December 9, 2015

1 Introduction

The keeper of a robot soccer team is an important component to the overall team. One task required of keepers is to execute save motions when a shot is underway in an attempt to prevent the ball from crossing the goal plane. Failure to detect a shot on goal could mean an uncontested goal for the other team, while falsely determining a shot on goal could result in an opportunity for the attacking player to dribble around the goalie and score. Developing a keeper behavior capable of reliably determining when a shot is underways is the focus of this project.

Reference 1

To this end, I took a data-driven machine learning approach to this problem. Specifically I trained discriminative models this task on data generated in simulation.

The outline of the this report will be as follows. First, I will formalize the problem and describe the main components of my approach. Next, I will discuss how my approach differs from keeper behaviors in past robocup tournaments. After that, I will discuss the process of generating a training set of shots via simulation, the process of training discriminative models, and evaluation both in simulation and on an Aldebran Nao robot. Finally I will conclude with future work and limitations of my approach.

I will refer to my nao robot as Cheesy throughout.

2 Approach

The problem at hand is to develop a keeper behavior capable of determining whether a shot is underway. Formally, I treat the problem of shot detection as a supervised learning classification, where the task is to find a function f:X -> Y where X = Powerset([x, y, dx, dy, distance, bearing]) and Y = {no-shot, left-shot, middle-shot, right-shot}. Here is a description of each of these attributes:

- (x, y): the position of ball in global coordinates
- (dx, dy): the velocity of the ball in global coordinates
- (distance, bearing): the distance and bearing of the ball, relative to Cheesy

Position and velocity estimates are derived via a Kalman filter and the distance and bearing are calculated from the ball detection code in our vision module. Position and velocity are in global "top-down" coordinates (see section WORLD) and distance and bearing are in Cheesy's local "egocentric" coordinate frame.

In this project, I explore using different feature spaces for training. Specifically I look at using [x, y, dx, dy], [x, y], and [distance, bearing].

3 Related Work

For the task of shot detection, the keeper generally has to model the game ball. Previous approaches by past robocup teams typically include modelling the velocity of the ball. In [7], the team models velocity by populating a queue with the past several global positional estimates of the ball. The queue is then split in half. The two halves are averaged, resulting in two average positions. These two positions

*ebanner@cs.utexas.edu

are then used to calculate a velocity vector in global coordinates. My approach is similar in that I also explicitly model the velocity of the ball, but with a Kalman filter instead.

Another approach to modeling the ball involves the use of a Kalman Filter [8], where the a Kalman filter is maintained for each distinct hypothesis regarding where the ball currently is. The team in [8] encodes their prediction model with 2D kinematics, relating predicted position of the ball at the next time step to its estimated velocity and position at the previous time step, as well as the friction between the ball and the floor [5]. In my approach, I do not incorporate friction into the estimates for position, but I do use a Kalman filter for modelling the ball.

One issue that arises with Kalman filters is that there are several parameters that need to be tuned by hand. In [1], the authors train discriminative models to learn these parameters, circumventing this issue. They induce supervision by placing a highly accurate GPS sensor on the robot whose position they are estimating. I did not pursue this approach because I did not have a similar way to induce supervision in a similar way.

Once a model of the ball has been obtained, it can be used as input to a shot detection algorithm. In [7], the team uses velocity of the ball to drive their decision whether to step to the goalie left, right, or execute a save (although they do not explicitly specify how this is done). In [6], the team simply executes a save motion when the ball gets within a certain distance. The team in [3] forgoes discrete "save" motions altogether, rather opting for an active approach that includes pursuing the ball in hopes of intercepting it. My approach is most similar to [7] in that the models I trained will likely predict shot if that ball is close, but my models have the ability of predicting no-shot when the ball is very close, but not moving.

The process of learning has been explored in robot soccer. For example, the team in [5] learns to model the friction between the floor and the ball by rolling the ball in front of the robot several times and calculating a parameter for this. In my approach, I do not learn the friction of the surface, but rather hard-code it into my algorithm for generating shots in simulation. Another example is the team in [4], in which they use reinforcement learning and a policy gradient method, in addition to genetic algorithms to optimize dive times. Unfortunately they do not go into details and it is also not clear whether they are using these learning approaches for optimizing how fast they can dive or when to decide to dive.

Learning from simulation has been explored in the robot soccer domain. [2] learns parameters for a robotic walk in simulation, but does not use these learned parameters on physical robots. In my approach, I learn shot detection parameters in simulation and apply them directly to Cheesy.

4 Methodology

In this section, I will detail the components relating to the end-to-end task of generating a good keeper behavior. I will start by discussing data generation process. Then I will discuss the models that train on this data. Finally, I will discuss evaluation on synthetic data, as well as on Cheesy.

4.1 The World

The first step in a supervised learning problem is to secure training data. I forgoed the idea of collecting data on Cheesy and chose instead to generate shots in simulation.

In order to generate shots which closely matched what Cheesy would experience, I used the Goalie Simulation tool Jake Menashe had developed for our use during the Kalman Filter assignment. I this simulation, when a shot is fired an initial velocity is set on the ball and that velocity decays by a constant factor each time step. Following the parameters that were used in the goalie simulator, I used a decay of 0.966 and 150 time steps for a shot. This corresponds to 3 seconds, since the vision code on Cheesy runs at 50 Hz. Figure JAKE_{SIM} provides a visual of the goalie simulation tool.

The green rectangle is 2,000 units tall and 3,000 units wide. 1 unit corresponds to 1mm in physical space. This field corresponds exactly to the miniature field in the robot soccer lab.

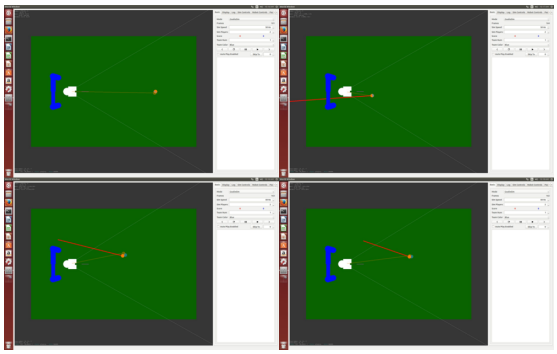


Figure 1: Visual of the goalie simulation tool we used in the Kalman filter assignment

4.2 Data Generation

Because I wanted to be able to generate many shots in a systematic way, I wrote my own version of the goalie simulator. Figure SHOT-PIPELINE shows my approach to generating batches of shots via simulation.

Unlike the in the goalie simulator, my goal is half the height of the field. This was the width given to the goal by our keeper behavior in the Kalman filter assignment. Since I wanted to use this behavior as a baseline, it made sense to encode the same width for both behaviors.

To generate kicks, two grids are drawn on the field; one in front of the goal and one around the goal. Shots were generated by choosing a point in each of the two grids and solving for the initial velocity required to have the ball land on the latter point after 150 time steps (with the decay factor). A kick is deemed a shot on goal if it crosses the goal plane before 150 time steps. Further, the regions within the goal are divided up into left, middle and right. The exact values for these boundaries once again came from our kalman filter keeper behavior in order to maintain consistency with the baseline. A small amount of gaussian random noise is applied to the position component of each kick data point. In general, I found that increasing the noise gives greater coverage of the field, with the expense of decreased overall accuracy.

After the kicks are generated, the kick points points behind the goal are filtered away because they cannot be sensed by Cheesy. Finally, shots into the left and right portion of the goal are undersampled until they match the number of middle shots. Likewise, misses are undersampled until the number of datapoints for misses matches the total number of shot datapoints. This was the simplest way to correct for class imbalance, which would hinder model performance if not corrected for. Figure $\text{TRAINING}_{\text{DATA}}$ gives a visual of the total number of kick points for each type of kick.

4.3 Feature Sets

My models trained on the following feature sets:

- Global position and velocity
- Global position
- Distance and Bearing

The motivation for training on position and velocity is because that is the data the kalman filter keeper uses for its shot detection algorithm. The models trained on position and distance and bearing are given the past 10 observations of each respectively. If the ball is not seen for five consecutive frames, then the window is cleared.

4.4 Models

The models I used for training include softmax, fully-connected neural network, and recurrent neural network classifiers. Each model uses a cross-entropy loss function. The softmax and fully-connected neural networks are trained with backpropagation on full batches of the training input. The recurrent neural network is trained with backpropagation through time with a rollout of 10. The fully-connected neural network has a one hidden layer of size 5 with sigmoid activation units and the RNN has one hidden state vector with size 30 and uses tanh nonlinearities. Each model was trained with a regularization term of 0.001. The softmax and FCNN used learning rates of 0.1 and the RNN used a learning rate of 0.005.

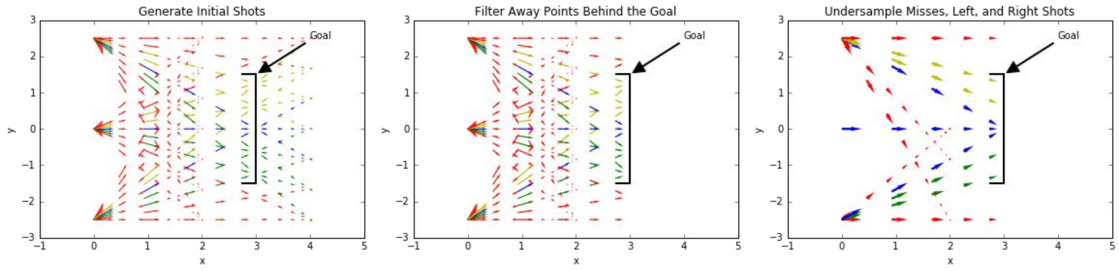


Figure 2: Example of the entire kick-generation pipeline. First, kick are generated. Next, kick points behind the goal are filtered away. Finally, shots to the left and right portions of the goal, as well as misses are undersampled to achieve an identical number of data points for each kind of kick. A time step value of 10 and a decay of 0.8 was used to generate these plots

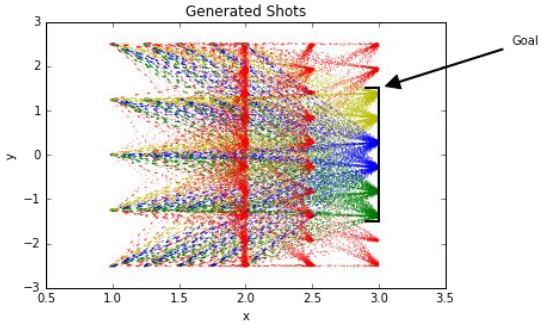


Figure 3: Full set of generated kicks. The class breakdown is as follows: 27230 data points for missed shots, 9039 for shots to the left, 9112 for shots to the middle, and 9017 for shots to the right.

Table 1: Training accuracies of classifiers on synthetic data. Raw accuracy is how many times the classifier predicted the correct class. Shot accuracy counts the classifier's prediction as correct if it predicts shot on a shot (e.g. predicting shot-middle when the correct label was shot-left).

	PV	PV	P	P	DB	DB
Raw Accuracy	Raw	Shot	Raw	Shot	Raw	Shot
SM	0.547	0.547	0.616	0.648	0.774	0.857
FCNN	0.703	0.759	0.846	0.884	0.884	0.940
RNN*	0.685	0.736	0.521	0.632	0.864	0.891

4.5 Simulation Evaluation

Each of the models was tested on the data it was trained on. Here are the accuracies.

Table SIM_{ACC} details the performance of each classifier on the training data. A separate cross-validation/test set was not held out because of the closed-world assumption, the idea being that the shots seen during test time would be very similar.

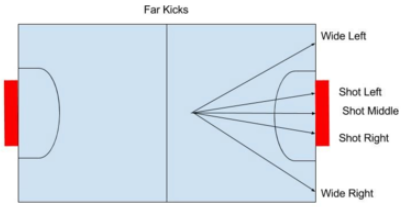
The FCNN consistently outperforms the softmax classifier. The SM classifier is only able to make coarse predictions, whereas the FCNN is able to make finer-grained decisions (e.g. a ball that starts out left will not necessarily end up left). The position and distance and bearing feature spaces increase performance primarily due to their input being windowed.

The recurrent neural network has an asterisk next to its name because it was trained on a dataset which was an order of magnitude smaller than the dataset the SM and FCNN classifiers were trained on due to the increased demands of RNN on training data. The training set it was trained on consisted of only 1,000 data points and each shot only consisted of 10 time steps. This drastically reduced the complexity of the domain and sped up training time substantially.

4.6 Cheesy Evaluation

After training each of the models, I implemented of them on Cheesy and conducted and evaluated each model and feature space combination on the real soc-

cer field. Figure $\text{REAL}_{\text{FIELD}}$ shows pictorially what each kick looked like.



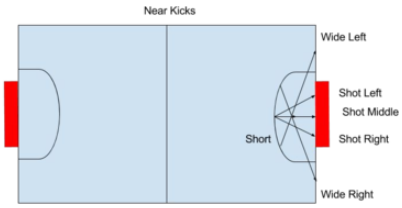


Figure 4: The shots used to evaluate performance of keeper behaviors. Note there is a short kick in the "near" kicks, but no short kick among the "far" kicks.

I kicked the ball five times for every kick among the far and short kicks illustrated in figure $\text{REAL}_{\text{FIELD}}$. For each behavior, I plotted the confusion matrices. I chose not to perform a short kick among the far kicks because it was too difficult to kick the ball sufficiently close to Cheesy consistently. Additionally, I consider the keeper has detected a wide shot correctly if it does nothing at all (e.g. no false positive). Figure $\text{KEEPER}_{\text{CONFUSION}}$ contains the confusion matrices for the kalman keeper.

The kalman keeper does very well on long shots. However, it struggles significantly on the shots close

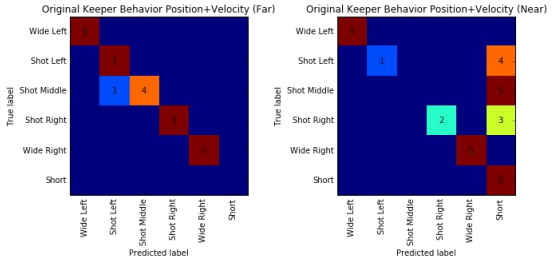


Figure 5: Confusion matrices for our original kalman filter keeper behavior. Note there is no short kick among the far kicks; it is only included to maintain symmetry with the near kicks. Numbers on the off-diagonal indicate misclassification

up. To be fair, the fact it only predicts shot after deciding shot for several frames puts it at a disadvantage, but in theory it should be able to classify these correctly.

Figure $\text{SOFTMAX}_{\text{CONFUSION}}$ show the confusion matrices for the trained softmax classifiers on each of the input feature spaces next to its predictions on the synthetic training data generated via simulation.

The position and velocity model learns to predict shot if the ball gets close enough to the goal along the x-axis. It reports many false positives because of this. The position model improves upon the position and velocity model, but struggles with false positives to the right. The distance and bearing model struggles with near shots because of its large region for middle shots extending outwards from the center of the goal.

Figure $\text{NN}_{\text{CONFUSION}}$ show the confusion matrices for the trained FCNN classifiers on each of the input feature spaces next to its predictions on the synthetic training data generated via simulation.

The position and velocity model learns a similar decision boundary to the softmax classifier, but is capable of detecting wide shots. The position model learns a good model of the most types of kicks. The distance and bearing model is unable to discriminate between shots up close, but does a good job at long distances.

Table $\text{BEHAVIOR}_{\text{ACCS}}$ lists the accuracies for each keeper behavior.

Overall, the best behaviors are the kalman keeper

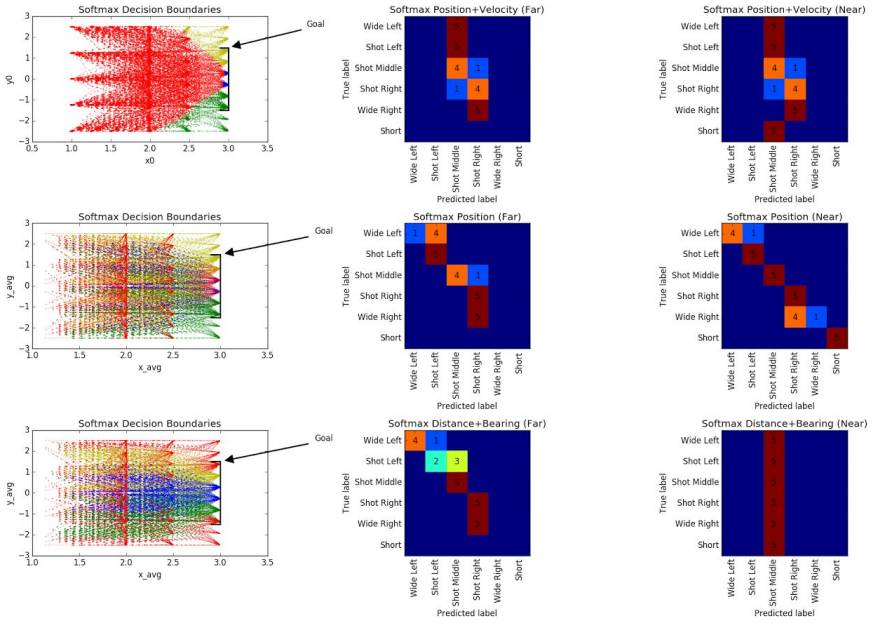


Figure 6: Predictions on synthetic data generated via simulation and confusion matrices during physical evaluation for softmax classifier

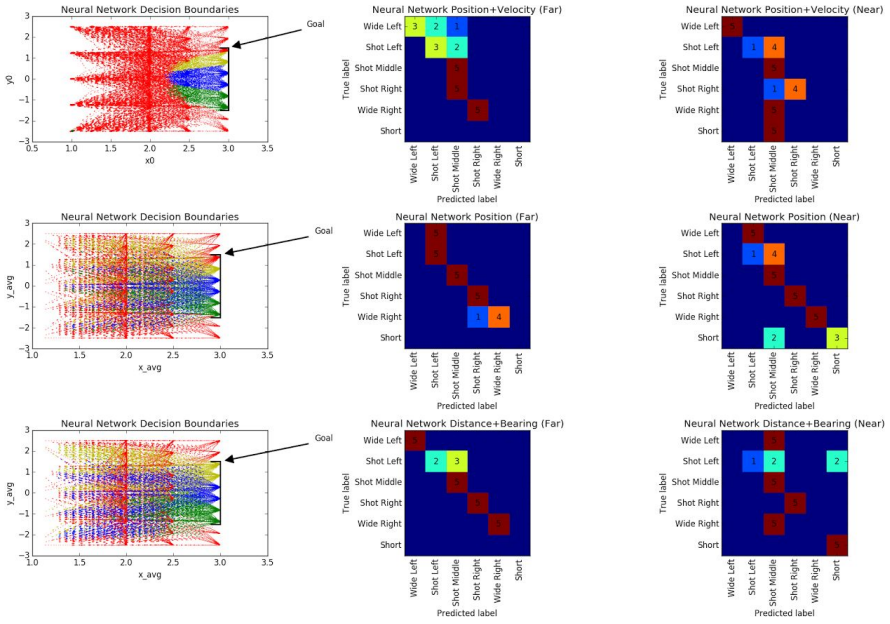


Figure 7: Predictions on synthetic data generated via simulation and confusion matrices during physical evaluation for FCNN classifier

Table 2: Training accuracies of behaviors on during shot task. Raw accuracy is how many times the behavior predicted the correct class. Shot accuracy counts the behavior's prediction as correct if it predicts shot on a shot (e.g. predicting shot-middle when the correct label was shot-left).

	Far		Close		Total	
	Raw	Shot	Raw	Shot	Raw	Shot
Kalman	0.960	1.000	0.600	0.600	0.764	0.782
SM PV	0.320	0.600	0.267	0.500	0.291	0.545
NN PV	0.440	0.720	0.500	0.667	0.473	0.691
SM P	0.600	0.640	0.833	0.833	0.727	0.745
NN P	0.760	0.760	0.633	0.767	0.691	0.764
SM DB	0.840	0.960	0.167	0.500	0.473	0.709
NN DB	0.880	1.000	0.533	0.600	0.691	0.782

and the FCNN trained on distance and bearing. The position models are most consistent between long and short shots.

The distance and bearing models' performance on near shots was the biggest disappointment. During my class demo, it looked as though the distance and bearing models were performing the best. But that was when the models were not differentiating between left, middle, and right shots.

There are a few surprises here. Even though the softmax position model was far less accurate than the neural network (table SIM_{ACC}) in simulation, it does better in the kicking task. This could be due to chance or it could be due to the fact that neural network was overfitting and not able to learn general enough decision boundaries.

The neural network trained on distance and vision bearing does well at long distances, but poorer at short distances. This is surprising since distance and bearing readings will be noisier at longer distances due to noise in Cheesy's camera.

It can be observed that the position models perform well at detecting close shots and the distance and bearing models do much better at longer distances. Ensembling these two methods has the promise of substantially outperforming the original keeper behavior overall. A simple weighting scheme where the distance and bearing prediction is weighted proportionally with the ball distance seems like a reasonable thing to do.

In order to increase performance with these models, it is tempting to generate more data. But the more data points around the center of the goal, the more difficult it becomes to discriminate between them. This could be mitigated by increasing the window size, but this comes at a cost as it takes longer to fill up the window, which may result in a prediction of shot after the ball has already gone by.

4.7 False Positives

In addition to accuracies, I recorded the number of false positives for each model. I defined a false positive as a prediction of shot in between kicks. Cheesy had a chance to predict shot as I was rolling the ball backwards to set up for the next kick and also when the ball was sitting still, waiting to be kicked. Figure FPS shows the number of false positive for each behavior.

The original behavior has no false positives because it never detects shots at close range. The models trained on position and velocity have a high number of false positives because of the noisy velocity estimates encountered during training for real. The position models do not have as many false positives and the distance and bearing models are somewhere in between.

4.8 Recurrent Neural Network

As stated in table SIM_{ACC}, the recurrent neural network was trained on a much smaller training set than

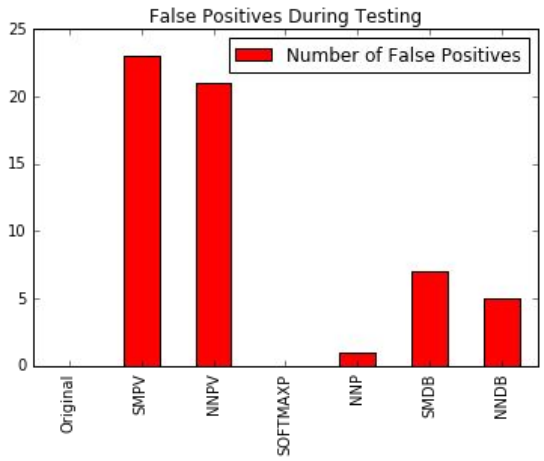


Figure 8: The number of false positives emitted by each behavior in between shots

the softmax and neural network classifiers. Unfortunately, this smaller training set did not result in a usable goalie behavior on Cheesy. Figure RNN shows the kick data the RNN was trained on, as well as the predictions for each of the feature spaces.

The RNN is a very appealing model for a keeper behavior. It does not require windowed input (unlike the softmax and FCNN keepers). Given more time to train and adding an additional hidden layer in front of the softmax layer, I am confident that it would achieve performance comparable to the softmax and FCNN behaviors.

5 Limitations

All of the learned behaviors make the assumption the keeper remains in a static position in the middle of the goal. In practice, it is often favorable for the keeper to take a more active approach and head off the attacker before the attacker can get into a good striking position.

6 Future Work

The obvious future work would be to train these models for longer periods of time and do a formal hyperparameter search via grid search, or some other method. Additionally using more trianing data would most likely help to improve performance. Additionally, noise is a parameter that can be tweaked that may have interesting affects.

Along a more interesting line, the idea of using less and less hand-crafted features is an appealing direction for future work. Even using ball distance and bearing introduces substantial noise that may prohibit the a keeper behavior from learning robust shot detection. An exciting next step forward is to remove ball distance and bearing features and rely solely on frame-space features such as the location of the ball in Cheesy's frame view along with the joint angles of Cheesy's head. The final step would be foregoing even these features and relying solely on pixels for features.

7 Conclusion

I implemented keeper behaviors that learn how to detect shots on goal. I generated shots in simulation, used this data to train discriminative models, implemented these models on Cheesy, and evaluated them against the meticulously hand-crafted kalman keeper behavior. I found models trained on position and ball distance and bearing features alone to be competitive with the kalman keeper behavior. Such work is one step towards moving away from hand-crafting behaviors and taking more data-driven machine learning approaches in the robot soccer domain.

References

[1] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y Ng, and Sebastian Thrun. Discriminative training of kalman filters. In *Robotics: Science and Systems*, pages 289–296, 2005.

[2] Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. Humanoid robots learning to

walk faster: From the real world to simulation and back. In *Proceedings of the 2013 international conference on autonomous agents and multi-agent systems*, pages 39–46. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[3] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, David Claridge, Hung Nguyen, Jayen Ashar, Stuart Robinson, and Yanjin Zhu. runswift team report 2010 robocup standard platform league. *University of New South Wales, Australia*, 2010.

[4] F Riccio, F Patota, F Bella, E Borzi, D De Simone, V Suriani, L Iocchi, and D Nardi. Spqr robocup 2015 standard platform league team description paper. 2015.

[5] Thomas Röfer, Tim Laue, Judith Müller, Michel Bartsch, Malte Jonas Batram, Arne Böckmann, Martin Böschen, Martin Kroker, Florian Maaß, Thomas Münder, et al. B-human team report and code release (2013), 2010.

[6] Peter Stone, Kurt Dresner, Selim T Erdogan, Peggy Fidelman, Nicholas K Jong, Nate Kohl, Gregory Kuhlmann, Ellie Lin, Mohan Sridharan, Daniel Stronger, et al. The ut austin villa 2003 four-legged team. In *in RoboCup-2003: Robot Soccer World Cup VII*. Citeseer, 2004.

[7] Peter Stone, Kurt Dresner, Peggy Fidelman, Nicholas K Jong, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. *The UT Austin Villa 2004 RoboCup four-legged team: Coming of age*. Computer Science Department, University of Texas at Austin, 2004.

[8] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

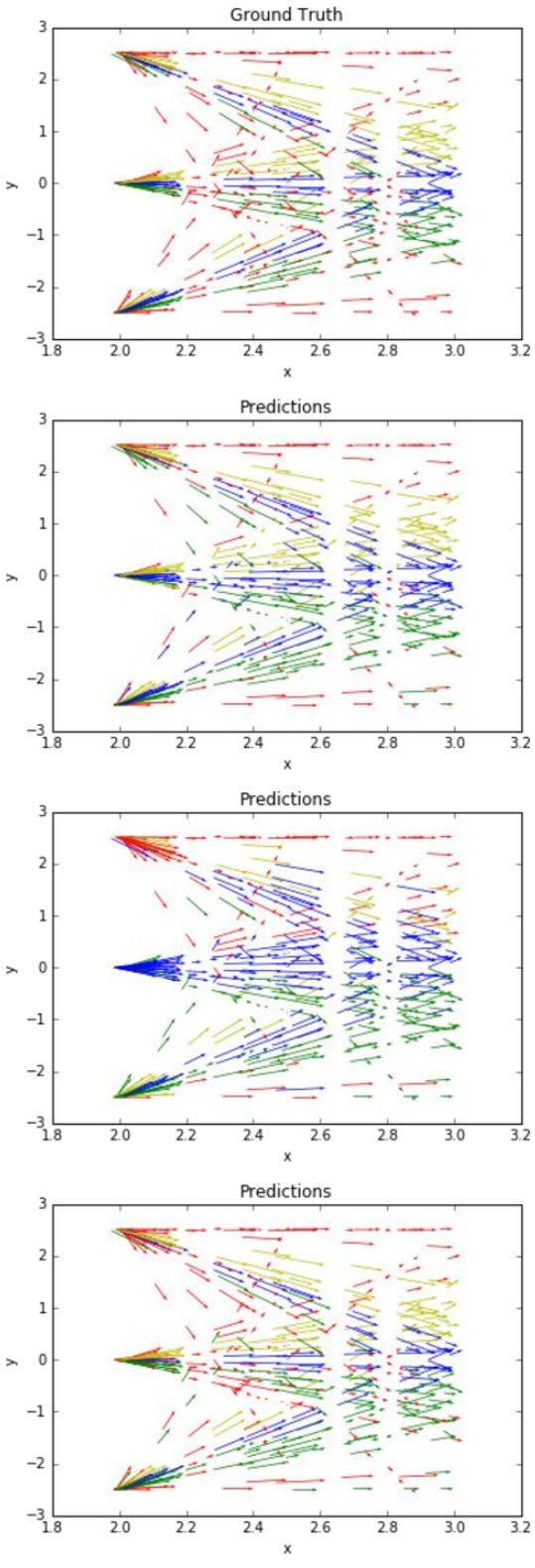


Figure 9: Predictions for the RNN trained on Position+Velocity, Position, and Distance+Bearing, from left to right.