The UPennalizers RoboCup 2015 Standard Platform League Team Description Paper

Yongbo Qian, Yizheng He, Qiao (Rachel) Han,
Sagar Poudel, Austin Small, Kyu Il Lee and
Dr. Daniel Lee

General Robotics Automation, Sensing and Perception (GRASP) Laboratory
University of Pennsylvania

Abstract

This paper presents the current research and software system developed on Aldebaran Nao robots by the UPennalizers - University of Pennsylvania Robot Soccer Team. The team has been involved in RoboCup for over a decade to foster the research in the areas of computer vision, machine learning, motion control and artificial intelligence. The current software system integrates perception, locomotion and behavior modules together to enable robots play soccer autonomously. The perception module detects landmarks on the soccer field and utilizes them to localize the robot. The locomotion engine allows omni-directional motions and uses sensory feedback to compensate for external disturbances. High-level behavior module uses Finite State Machines to define single robot's behavior as well as team coordination. The work also includes the improvements made across all modules to address the significant environmental changes from previous years in the RoboCup 2015 competition in Hefei.

1 Introduction

The UPennalizers is affiliated with General Robotics, Automation, Sensing and Perception (GRASP) Laboratory and the School of Engineering and Applied Science at University of Pennsylvania. In 1999, two years after the first international RoboCup, this robot soccer team was formed and began stepping up to the challenges put forth by the competition. While the league was still utilizing four-legged Sony Aibos, the UPennalizers made the quarterfinal rounds every year through 2006 before taking a brief two-year hiatus in 2007. The team reformed and returned in 2009 to begin competing in the Standard Platform League with Aldebaran Nao robots, taking on bipedal motion alongside improved vision techniques and advanced behavior control. In 2015, with most of the team members graduated, a relatively new team was formed to take on more challenges in recognizing white goal posts, designing coaching robot behaviors as well as improving the localization and locomotion systems. Coached by Dr.Daniel Lee, the 2015 team consists of Yizheng (Dickens) He, Yongbo Qian, Sagar Poudel, Kyu Il Lee, Qiao (Rachel) Han and Austin Small.

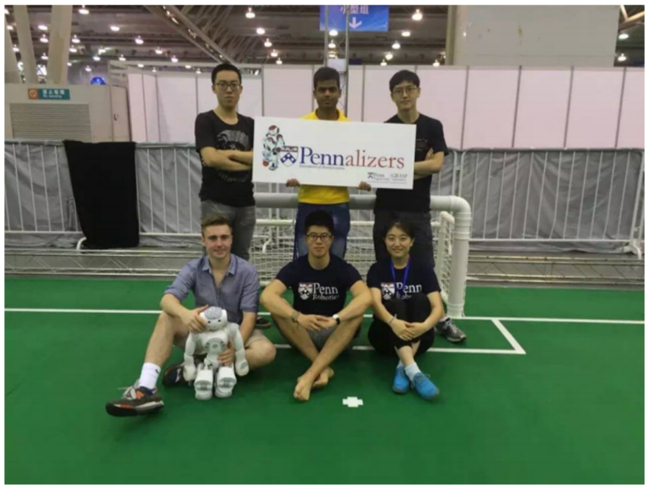


Fig. 1. The 2015 UPennalizers Team. Top from left to right: Yizheng He, Sagar Poudel, Kyu Il Lee. Bottom from left to right: Austin Small, Yongbo Qian, Qiao Han

2 Software Architecture

A high-level description of the software architecture for the Nao robots is shown in Figure 2. The current architecture is an expansion built upon the previous years work. It uses programming language lua as a common development platform to interface between all modules.

The system maintains a constant update speed of 100Hz, and is decoupled into two separate pipelines. The main process handles motion control and behavior control, while an auxiliary process is dedicated solely to perception processing. This decision allows for more efficient handling of the Nao's on-board single-core Intel Atom Z530 clocked at 1.6 GHz. The perception module runs upon the remaining processing power not used by the main modules, and as a result, the Nao robots were noted to be much more stable and robust than in previous years.

Low-level interactions with hardware are implemented using compiled C++ libraries in conjunction with the Nao's on-board hardware controller (NaoQi) or custom controllers. These in turn, are called via Lua scripts, and allow for control over motor positions, motor stiffnesses, and LED's. Sensory feedback is also handled similarly, allowing users to get data from a variety of sources such as the Nao's two on-board cameras, foot weight sensors, the inertial measurement unit (IMU), and the ultrasound microphones.

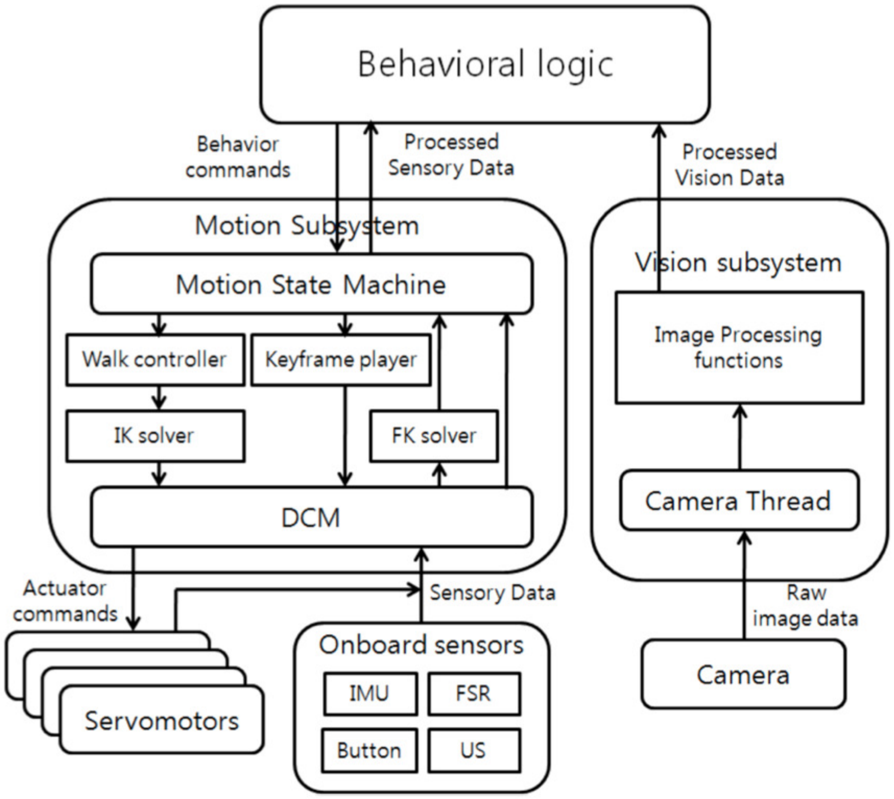


Fig. 2. Block Diagram of the Software Architecture.

Inter-process communication is accomplished via shared memory. Important information such as relative ball distance, robot's position on the field, and game state are stored in the shared memory and can be read or written by any module. In addition, any operator connected to a Nao via secure shell can monitor the data stored in the shared memory module without any change or impact on the running system, allowing for real-time on-the-fly debugging and analysis through either Lua or MATLAB.

The main modules accessed by our Lua routines are as follows, layered hierarchically:

Camera Direct interface to the cameras located in the forehead (upper) and mouth (lower); controls switching frequency and bundling of images in YUYV format.

Vision Interprets incoming images; based on the user-created color table and camera parameters, the module passes on information relating to the presence and relatively location of key objects such as the ball, defending goal posts, attacking goal posts, field lines, field corners, center circle and other robots.

World Models the robot's state on the field, including pose state and filtered ball position;

Body Handles physical sensory information and functions; reads joint encoders, IMU data, foot weight sensors, battery voltage, and chest button presses, but can also set motor positions, stiffnesses, and LED's.

Motion Dictates general movements on the Nao; i.e. sitting, standing, diving

Walk Controls omni-directional locomotion; takes in hand-tuned parameters and applies them to a zero-moment point (ZMP) based walk engine.

Kick Maintains intra-robot stability during kick movements; different kick settings can be loaded to allow for powerful standing kicks, quick walk-kicks, and decisive side-kicks.

Keyframes Lists scripted positions for certain movements; getting up from front and back falls is done by feeding the Body module a series of motor positions and timings.

Game State Machine Receives and relays information from the Game Controller; information from the GSM such as game state determines behavior among all robots on the field during certain times of the game.

Head State Machine Controls head movements; different conditions determine when to switch into ball searching, ball tracking, and simply looking around.

Body State Machine Dispatches movement instructions; conditions from all previous modules will cause the Nao to switch between chasing after far away balls, performing curved approaches to line up for shots, dribbling, and performing kicks when the ball is close enough.

3 Perception

Our robots divide the complex task of perception into different sub-tasks which are pieced together at run-time in an appropriate manner. Figure 3 shows the framework of the perception module.

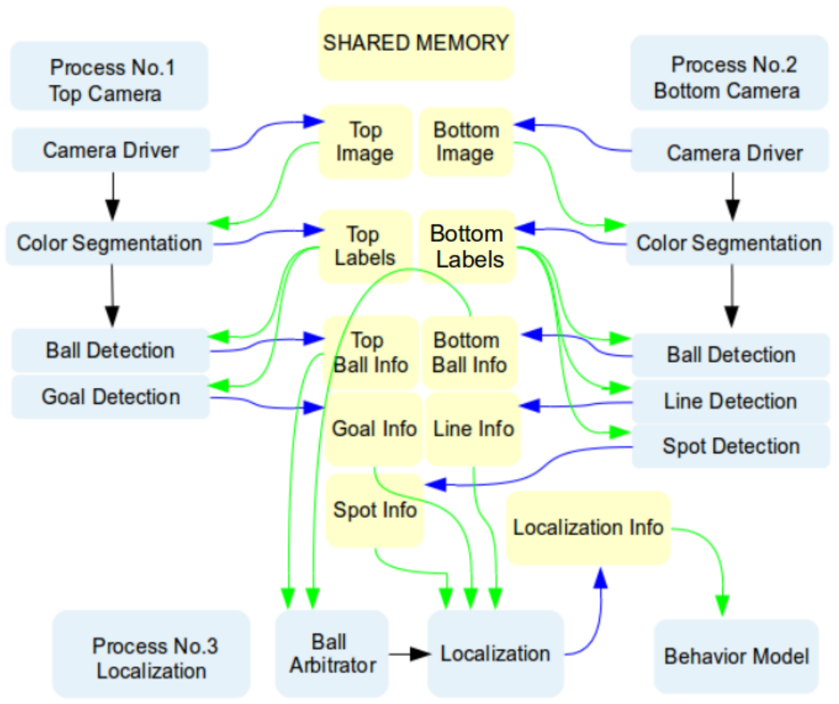
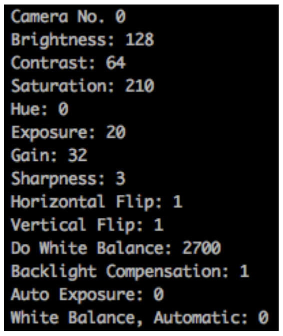


Fig. 3. Block Diagram of the Cognition Module.

Each Aldebaran Nao robot has two cameras on board, also known as the top (forehead) and bottom (mouth) camera. They feed images in stream at 30 frames per second each. Images from the top and bottom cameras are processed in parallel by two processes. Different object detection routines are performed on images obtained from different cameras to improve accuracy and efficiency. An arbitrator process coordinates intermediate results from the two processes and runs the localization module. The perception module outputs robots positions on the field and relative positions of objects of interest to each robot and sent them to behavior controls.

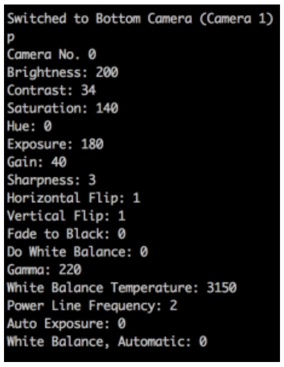
3.1 Camera Parameters

Since object detection depends highly on the quality of images from the cameras, tuning the camera parameters (i.e. Exposure, Contrast, and Saturation) is an important task. A simple GUI is implemented allows users to easily change each parameter and check the image quality. Figures 4(a) and 4(b) are screen shots of the script. Note that NAO V5 robot contains some additional parameters which are useful.



```
Camera No. 0
Brightness: 128
Contrast: 64
Saturation: 210
Hue: 0
Exposure: 20
Gain: 32
Sharpness: 3
Horizontal Flip: 1
Vertical Flip: 1
Do White Balance: 2700
Backlight Compensation: 1
Auto Exposure: 0
White Balance, Automatic: 0
```

(a) NaoV4 Camera Parameters



```
Switched to Bottom Camera (Camera 1)
p
Camera No. 0
Brightness: 200
Contrast: 34
Saturation: 140
Hue: 0
Exposure: 180
Gain: 40
Sharpness: 3
Horizontal Flip: 1
Vertical Flip: 1
Fade to Black: 0
Do White Balance: 0
Gamma: 220
White Balance Temperature: 3150
Power Line Frequency: 2
Auto Exposure: 0
White Balance, Automatic: 0
```

(b) NaoV5 Camera Parameters

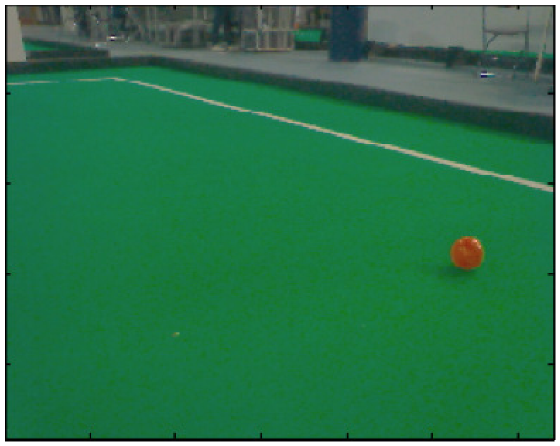
Fig. 4. Example Camera Parameters

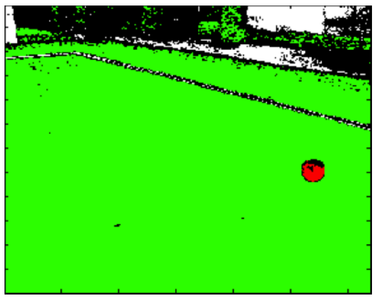
3.2 Color Segmentation

The raw input images are down-sampled and switched to YUV color space. A Gaussian Mixture Model(GMM) is then used to partition the color cube into six colors:

- Orange (Ball)
- Green (Field)
- White (Goal Posts, Lines)
- Pink (Robot Jerseys)
- Blue (Robot Jerseys)
- Black (Others)

The segmentation process follows a supervised learning routine: images taken from both cameras are trained to form a color look-up table. While the robot is running, the image processing pipelines segment raw images into discretely colored images by classifying individual pixels based upon their mapped values in the color table. The segmented images are further shrunk to lower-resolution colored labels by merging adjacent pixels to ensure the computational efficiency. Figure 5(a) to 5(c) show each step of color segmentation.


(a) Raw Image


(b) Segmented Image

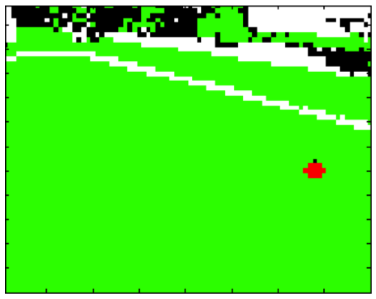

(c) Colored Label

Fig. 5

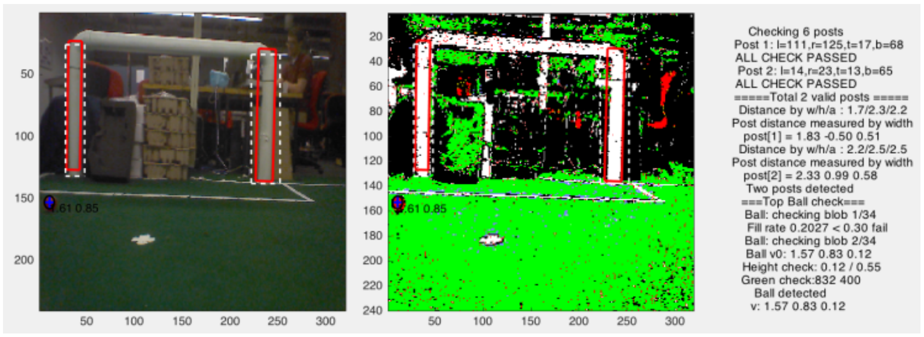
3.3 Object Detection

The next step is object detection on color-segmented images. Connected regions are recognized as either connected components or edge regions, and the object recognition is defined using bounding box and the centroid location. On top of the associated color features, different detection criteria are applied to each object.

One of the biggest rule changes in 2015 was the color of goalpost. Now with white goalposts, supporting structures and net, at well as white field lines, robots and massive environmental noise, this color feature became extremely unreliable for previous detection approaches. In order to handle this change, improved algorithms for goalpost detection and line detection were developed.

3.3.1 White Goalpost Detection

The improvements were made on further utilizing geometry features and exploiting field context. For geometry features, strict check on size, orientation, aspect ratio of the bounding box was performed to exclude some false positive goalposts from random white noise. For field context, a ground check for goalpost base and an ambient check for surrounding area were also implemented. This helps to distinguish goalposts from white field lines since the goalposts only vertically grow on the field, but not surrounded by field green. In addition, analysis of goalpost's centroid with robot's horizon was also performed to get rid of some false detections such as other robot's arms. Figure 6 shows the successful detection on white goalposts along with the debugging messages.



```
Checking 6 posts
Post 1: l=111,r=125,t=17,b=68
ALL CHECK PASSED
Post 2: l=14,r=23,t=13,b=65
ALL CHECK PASSED
=====Total 2 valid posts =====
Distance by w/h/a : 1.7/2.3/2.2
Post distance measured by width
post[1] = 1.83 -0.50 0.51
Distance by w/h/a : 2.2/2.5/2.5
Post distance measured by width
post[2] = 2.33 0.99 0.58
Two posts detected
===Top Ball check===
Ball: checking blob 1/34
Fill rate 0.2027 < 0.30 fail
Ball: checking blob 2/34
Ball v0: 1.57 0.83 0.12
Height check: 0.12 / 0.55
Green check:832 400
Ball detected
v: 1.57 0.83 0.12
```

Fig. 6. White Goalpost Detection

3.3.2 Field Line Detection

During previous years, a 1D Hough Transform was implemented to detect field lines. Although this method was proven to be accurate, when extending it to detect other field features like center circle, the algorithm requires the computation of matrix accumulation in higher dimension which is not efficient enough. In order to integrate the detection of all field features, a simple and fast approach was explored this year. The approach can be decomposed to four subtasks:

Firstly, all regions that are loosely line shaped are detected via a green - white - green state machine. This state machine filters out multiple line segments which themselves are white and border on green regions. Inside the middle white state, one transitional state for black pixel is allowed in order to be tolerant to noise. Width check and height check are then performed to exclude false positive line segments. The width of line is counted by the pixel number inside white state, which should be within certain range. The line segments should also have roughly the height that a field line would have at the current image location, computed through robot's head angle transformation.

Secondly, the line segments are combined using a simple greedy algorithm. For the neighboring line segments with roughly the same orientation, if most of pixels between two center points are white, those segments belong to the same field line and a straight line is then fit to connect them. The search continues to find all remaining fitting segments. The line parameters keep updating through this process.

Thirdly, the rest of the line segments are checked if they could form a circle. Since the circle always appears with the center line, the intersection between the normal of each line segments and the center line is calculated. If the standard deviation of those intersection points is close to zero, those line segments can be labeled as center circle, and the average of intersection points can be consider as the center of circle. Figure 7 plots out the center of circle calculated by those line segments on the center circle

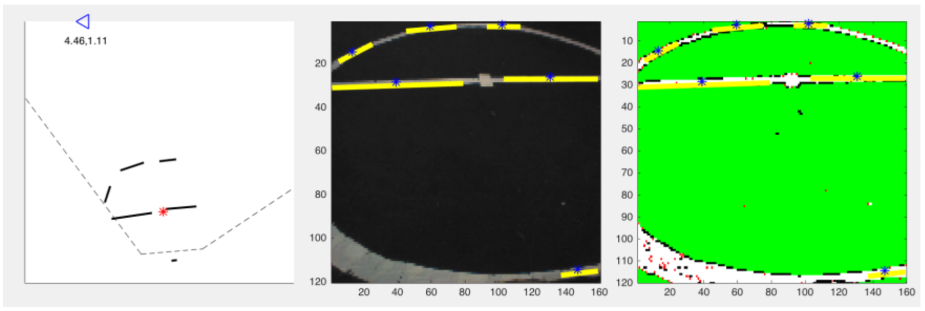


Fig. 7. Center Circle Detection

Finally, unlabeled or unconnected line segments are removed before performing the corner detection. If two field lines projected to the field intersect under a nearly perpendicular angle and the end point of both lines is close to the intersection, then those two lines are classified into one corner, which is shown in figure 8

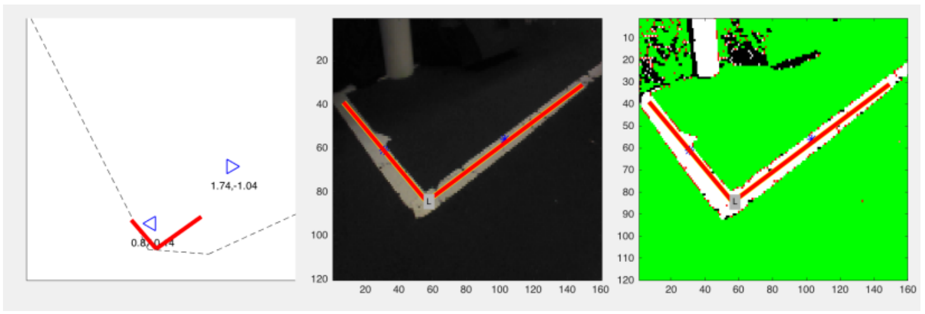


Fig. 8. Corner Detection

3.4 Self-Localization

The problem of knowing the location of robots on the field is handled by a probabilistic model incorporating information from visual landmarks such as goals and lines, as well as odometry information from the effectors. Recently, probabilistic models for pose estimation such as Extended Kalman Filters, grid-based Markov Models, and Monte Carlo Particle Filters have been successfully implemented. Unfortunately, complex probabilistic models can be difficult to implement in real-time due to a lack of processing power on robots. This issue is addressed with a pose estimation algorithm that incorporates a hybrid Rao-Blackwellized representation that reduces computational time, while still providing a high level of accuracy. The algorithm models the pose uncertainty as a distribution over a discrete set of heading angles and continuous translational coordinates. The distribution over poses (x, y, θ), where (x, y) are the two-dimensional translational coordinates of the robot on the field, and θ is the heading angle, is first generically decomposed into the product:

$$P(x, y, \theta) = P(\theta)P(x, y|\theta) = \sum_i P(\theta_i)P(x, y, |\theta_i) \tag{1}$$

The distribution $P(\theta)$ is modeled as a discrete set of weighted samples $\{\theta_i\}$, and the conditional likelihood $P(x, y|\theta)$ as simple two-dimensional Gaussian. This approach has the advantage of combining discrete Markov updates for the heading angle with Kalman filter updates for translational degrees of freedom.

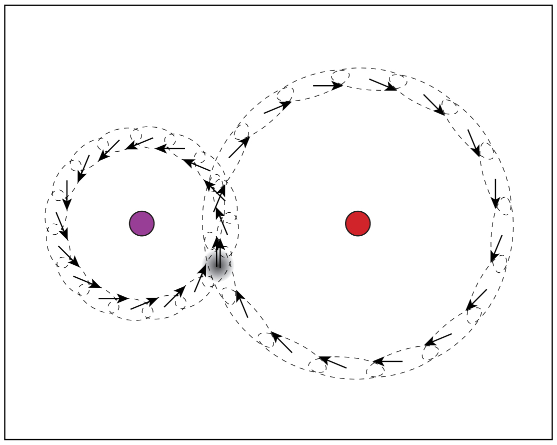


Fig. 9. Rao-Blackwellized probabilistic representation used for localization.

This algorithm enables robots to quickly incorporate visual landmarks and motion information to consistently estimate both the heading angle and translation coordinates on the field as shown in Figure 9. Even after the robots are lifted ('kidnapped') by the referees, they will quickly re-localize their positions when they see new visual cues.

3.4.1 Particle Initialization

The localization algorithm utilizes 200 particles to estimate the position of the robot. Properly initializing the positions of the particles helps improve the accuracy of the localization algorithm. Before the game starts, the particles are initialized on the sides of the defending half of the field, as shown in Figure 10. In the Set state, if the robot is not manually replaced, its particles are initialized near the possible initial positions defined in our game strategy. Besides, during the game, when a robot falls down, its localization particles' heading angles are reinitialized.

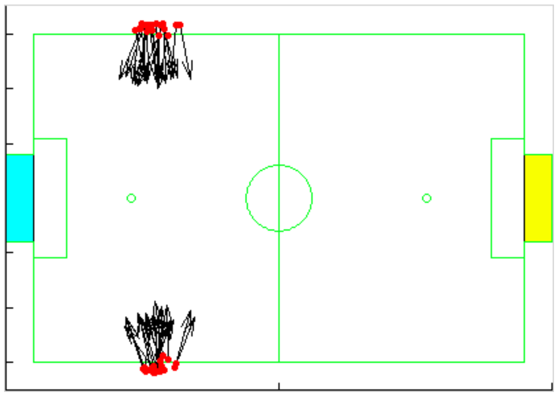


Fig. 10. Initialization of particles before game starts.

3.4.2 Odometry, Landmark Observation and Re-sampling

A Kalman filter is implemented to track the continuous change on the position and weight of each particle. The filtering is a product of two steps: the motion model update and the measurement update. The motion model update - also referred to as the odometry update - utilizes the robot kinematics to update the particle filter as the robot walks around the field. Given the joint angles of the robot, forward kinematics is used to compute the location of the robot's feet as it walks. The change in translation and rotation of the body of the robot are computed based on the position of the feet, as shown in Figure 11, and used to update the particle filter.

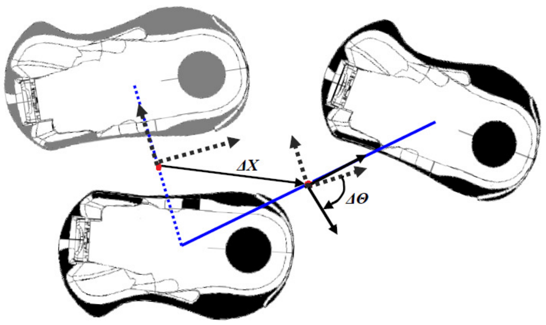


Fig. 11. Visualization of the odometry calculation after one step.

a set of camera parameter values. These parameters should make the top and bottom camera visually appear as similar as possible because both

The measurement model refines this estimate using sensory inputs, such as vision-based landmark detection. The measurement model incorporates positions of landmarks to adjust the particle positions and their weights in the filter. Positions of landmarks are weighted based upon their reliability. For instance, goal post detection is considered convincing and used to correct both the position and the direction of the robot. Meanwhile, line detections are only used to correct the direction due to large variance in their position calculation.

This year, the measurement model for goal posts, center circle and corners was further improved. A new triangulation method was implemented when both goal posts are detected. This method has a correction mechanism to correct the angle of further goal post based on closer goal post, and then use center of two posts to fix robot's orientation. Combining with goal post distance factor, the accurate robot position can be triangulated.

Since the detection of white goal posts are not as reliable as before, the weights of corner and circle are increased in the measurement model. The definitions of corner and circle were modified to become objects with orientation, so that they can also correct both position and direction of the robot. The position and angle of corner are defined to be the global coordinates of the vertex angle of the bisector, while the center circle has a position of the center point and an orientation of the center line.

The algorithm re-samples every 0.1 seconds. A stratification method is used to redraw all particles so that the ones with higher weight will stay. Figure 12 illustrates the result of self-localization.

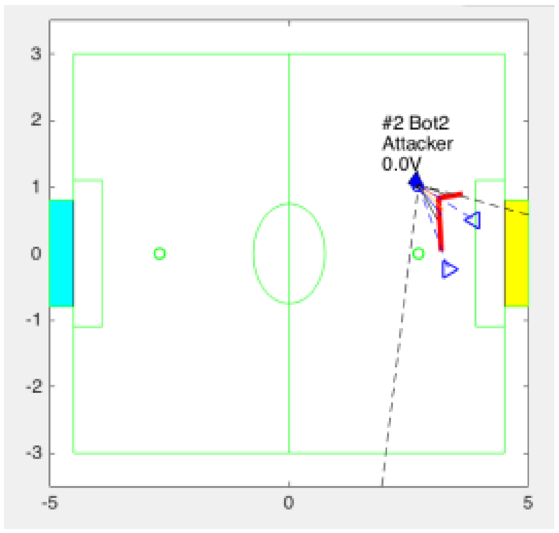


Fig. 12. The robot weighs corner and goalposts to establish an accurate estimation of its position and orientation on the field

3.4.3 Error Correction

One great challenge with self-localization in the Standard Platform League is the symmetric field. Under ideal circumstances where the robot's starting position is known, the basic particle filter approach alone is enough to keep track of the correct robot pose. However, noise in the motion model, inevitable false positive detections of landmarks, and falling down, will all eventually cause the robot to converge on a pose that is symmetrically opposite the true location. A team correcting mechanism based on goalie ball information is introduced.

Since the goalie mostly stays in penalty box and close to the defending goal posts, it should be more confident about the location of itself and the ball than other robots on field. The correcting mechanism works when a player robot and the goalie see the ball simultaneously but believe the ball is on different sides of the field. Under such circumstances, it is very likely that the player robot's localization is flipped and its particles will be flipped back against the center of the field.

This year, the coach robot was also implemented with similar error correction mechanism. Serving as a global observer, the coach robot constantly tracks the ball and can send human readable messages to the visualizers and flip back the flipping robot if needed. Although not being used during competition, the coach robot was working fine when testing in lab.

Moreover, robots are very likely to generate localization error when they fall over near the center of the field. The correcting mechanism labels these robots as "confused players", which will not make direct shots to goal. Instead, they will dribble or slightly kick the ball until the goalie sees the ball and confirms their positions.

4 Motion

Motion is controlled by a dynamic walk module combined with predetermined scripted motions. One main development has been a bipedal walk engine that allows for fast, omni-directional motions. IMU feedback is used to modulate the commanded joint angles and phase of the gait cycle to provide for further stability during locomotion. In this way, minor disturbances such as carpet imperfections and bumping into obstacles do not cause the robot to fall over. The robot has several powerful kick motions using pre-defined keyframes and a ZMP walk-kick for quicker reaction. We are also using keyframes for our get-up motions under different battery levels.

4.1 Walk

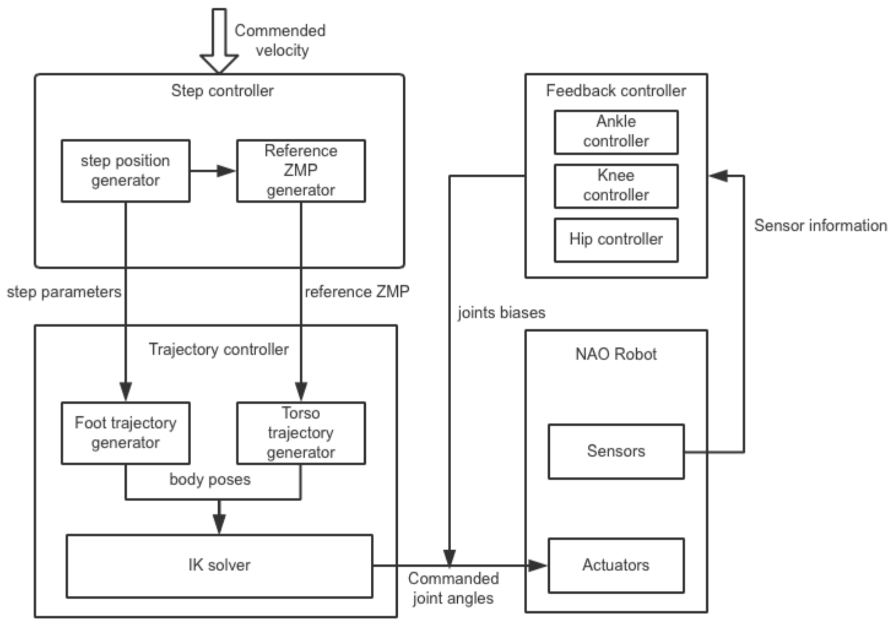


Fig. 13. Overview of the walk controller

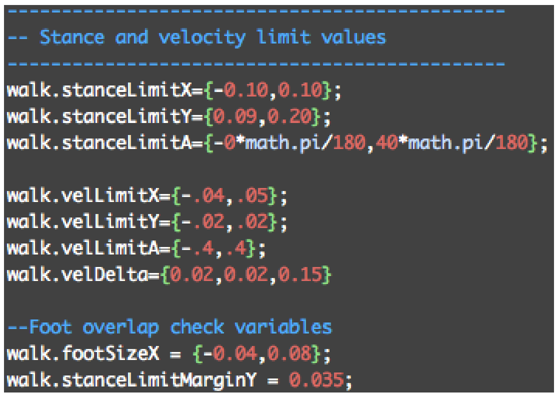
The walk engine generates trajectories for the robot's center of mass (COM) based upon desired translational and rotational velocity settings. The module then computes optimal foot placement given this desired body motion. Inverse kinematics (IK) are used to generate joint trajectories so that the zero moment point (ZMP) is over the support foot during the step. This process is repeated to generate alternate support and swing phases for both legs.

4.1.1 Step controller

The step controller determines the parameters of each step give different commended velocity and hardware parameters. Each step is defined as

$$STEP_i = \{SF, t_{step}, L_i, T_i, R_i, L_{i+1}, T_{i+1}, R_{i+1}\}$$

where SF denotes the support foot, t_{step} is the duration of the step, L_i, R_i, T_i and $L_{i+1}, R_{i+1}, T_{i+1}$ are the initial and final 2D poses of left foot, right foot and torso. L_i, R_i, T_i are the final poses from the last step, L_{i+1}, R_{i+1} are calculated using the commended velocity. Foot reachability and self-collision constraints are also considered given different configurations in the pre-defined walk setting file, as shown below.



```
------------------------------------------------
-- Stance and velocity limit values
------------------------------------------------
walk.stanceLimitX={-0.10,0.10};
walk.stanceLimitY={0.09,0.20};
walk.stanceLimitA={-0*math.pi/180,40*math.pi/180};

walk.velLimitX={-.04,.05};
walk.velLimitY={-.02,.02};
walk.velLimitA={-.4,.4};
walk.velDelta={0.02,0.02,0.15}

--Foot overlap check variables
walk.footSizeX = {-0.04,0.08};
walk.stanceLimitMarginY = 0.035;
```

Fig. 14. Example parameters for one of our walk files.

To get the most stable posture, the center of mass should lies on the middle point of two feet. Thus the target torso pose T_{i+1} is set to the midpoint of L_{i+1} and R_{i+1}. Given the initial and final position of the feet and torso, the reference ZMP trajectory $p_i(\phi)$ as the following piece-wise linear function for the left support case

$$p_i(\phi) = \begin{cases} T_i(1 - \frac{\phi}{\phi_1}) + L_i \frac{\phi}{\phi_1} & 0 \leq \phi < \phi_1 \\ L_i & \phi_1 \leq \phi < \phi_2 \\ T_i(1 - \frac{1-\phi}{1-\phi_2}) + L_i \frac{1-\phi}{1-\phi_2} & \phi_2 \leq \phi < 1 \end{cases}$$

where ϕ is the walk phase and ϕ_1, ϕ_2 are the timing parameters determining the duration of single support phase and double support phase.

4.1.2 Trajectory controller

The trajectory controller generates torso and foot trajectories for the current step. We first define ϕ_{single} as the single support walk phase

$$\phi_{single} = \begin{cases} 0 & 0 \leq \phi < \phi_1 \\ \frac{\phi - \phi_1}{\phi_2 - \phi_1} & \phi_1 \leq \phi < \phi_2 \\ 1 & \phi_2 \leq \phi < 1 \end{cases}$$

We then define a parameterized trajectory function

$$f_x(\phi) = \phi^\alpha + \beta\phi(1 - \phi)$$

to generate the foot trajectories

$$L_i(\phi_{single}) = L_{i+1} f_x(\phi_{single}) + L_i(1 - f_x(\phi_{single}))$$

$$R_i(\phi_{single}) = R_{i+1} f_x(\phi_{single}) + R_i(1 - f_x(\phi_{single}))$$

The torso trajectory $x_i(\phi)$ is calculated by modeling the robot as a inverted pendulum. Thus

$$p = x - t_{ZMP}\ddot{x}$$

With the reference ZMP trajectory we defined before, the $x_i(\phi)$ during the step with zero ZMP error should be

$$x_i(\phi) = \begin{cases} p_i(\phi) + a_i e^{\frac{\phi}{\phi_{ZMP}}} + b_i e^{-\frac{\phi}{\phi_{ZMP}}} - \phi_{ZMP} m_i sinh \frac{\phi - \phi_1}{\phi_{ZMP}} & 0 \leq \phi < \phi_1 \\ p_i(\phi) + a_i e^{\frac{\phi}{\phi_{ZMP}}} + b_i e^{-\frac{\phi}{\phi_{ZMP}}} & \phi_1 \leq \phi < \phi_2 \\ p_i(\phi) + a_i e^{\frac{\phi}{\phi_{ZMP}}} + b_i e^{-\frac{\phi}{\phi_{ZMP}}} - \phi_{ZMP} n_i sinh \frac{\phi - \phi_1}{\phi_{ZMP}} & \phi_2 \leq \phi < 1 \end{cases}$$

where $phi_{ZMP} = t_{ZMP}/t_{STEP}$. m_i, n_i are ZMP slopes defined as following for left support case

$$m_i = \frac{L_i - T_i}{\phi_1}, \;\; n_i = -\frac{L_i - T_{i+1}}{1 - \phi_2}$$

and for right support case

$$m_i = \frac{R_i - T_i}{\phi_1}, \;\; n_i = -\frac{R_i - T_{i+1}}{1 - \phi_2}$$

a_i, b_i can then be calculated given the boundary condition $x_i(0) = T_i$ and $x_i(1) = T_{i+1}$. The resulting ZMP and torso trajectory are shown below

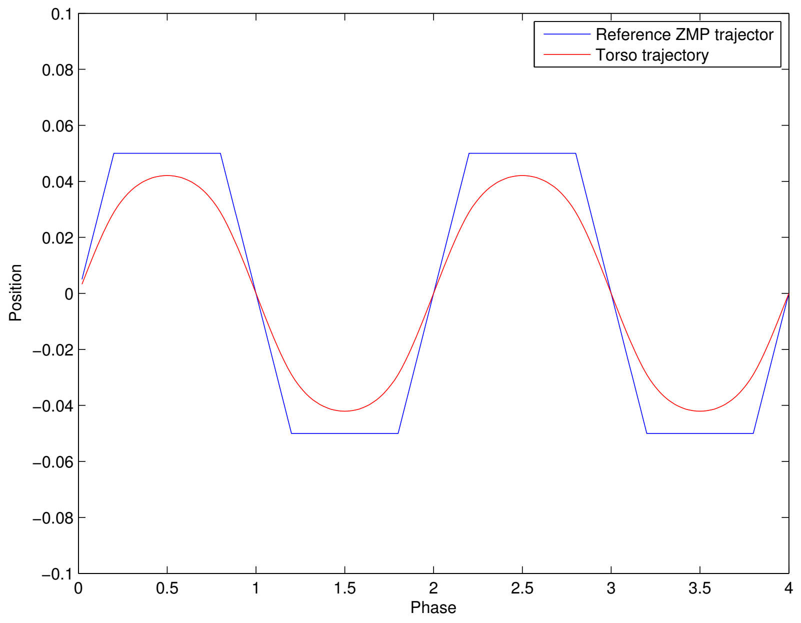


Fig. 15. An example for a reference ZMP trajectory and corresponding torso trajectory.

After we get the body pose and feet poses, the inverse kinematics module is used to calculate angles for each joints so that the robot can actually walk as we desired.

4.1.3 Feedback controller

The feedback controller takes in the sensor information from the robot during the walk and tries to stabilize it by controlling the ankle joints, knee joints and hip joints. Right now we are using the roll and pitch angles as inputs for several simple PD controllers. Knee and hip joints are used to overcome pitch errors while ankle joints are used for roll error. If the pitch angle or roll angle is higher than a threshold, the walk motion will be stopped. This can be caused by other robot pushing our robot during a game.

4.2 Kicks

Our kicks this year are a combination of scripted keyframes and ZMP-based kicks. Of our three kicks – standing, walk, and side – only the walk-kick utilizes the new ZMP engine. The old-fashioned style kicks are created by specifying motor positions and timings, and must be carefully tuned by hand in order to ensure balance, stability, and power. The new kicks are inherited from our merge with Team DARwIn. Similar to how the walk engine calculates joint positions in response to motion requests of the COM and ZMP, our newer kick calculates the way that the robot needs to balance in order to perform faster and more powerful kicks.

5 Behavior

Finite state machines (FSMs) dictate the behaviors and allow robots to adapt to constantly changing conditions on the field. The implementation of an FSM consists of a series state definitions and one arbitrator file that defines the transitions between states.

Each state consists of three stages: `entry`, `update` and `exit`. The `entry` and `exit` stages specify actions to be taken when the robot first enters a state or finally completes a state. During the `update` stage, which is in between, the robot constantly cycles through a routine. Usually it moves on the field while querying cognition information, until certain conditions are met. For instance, in state *BodySearch*, the robot rotates in place until either it detects the ball or it times out after not seeing the ball for 5 seconds. In the first case, it transits into a state of chasing the ball; in the other case, the robot transits into a state of going to the center of the field (where it has a better chance of detecting the ball).

5.1 The Body Finite State Machine

The Body Finite State Machine (BodyFSM) is the main behavior control module. It regulates robot movements and kicks during the game. Two sets of state machines are implemented, one for normal players and the other for the goalie. The key difference is that the goalie needs hold its position and stay in the penalty box in most cases while field players need to walk around and chase the ball.

5.1.1 BodyFSM for Normal Players

Figure 18 shows the transitions of states for a normal robot player. It is followed by brief descriptions of the states. The basic routine is: `Search for ball` $\rightarrow$ `Approach the ball` $\rightarrow$ `Dribble of Make a Shot`.

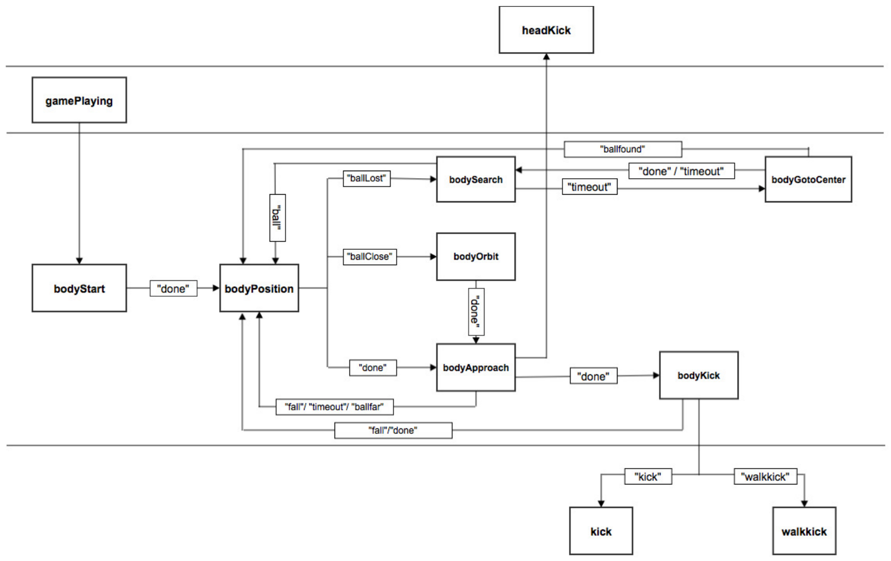


Fig. 16. Body State Machine for a non-goalie player.

bodyApproach Align for kick.

bodyGotoCenter Return to the center of the field.

bodyKick Perform different type of kicks.

bodyOrbit Make fine adjustments to trajectory before kicking.

bodyPosition Main body state; most states will transition back here.

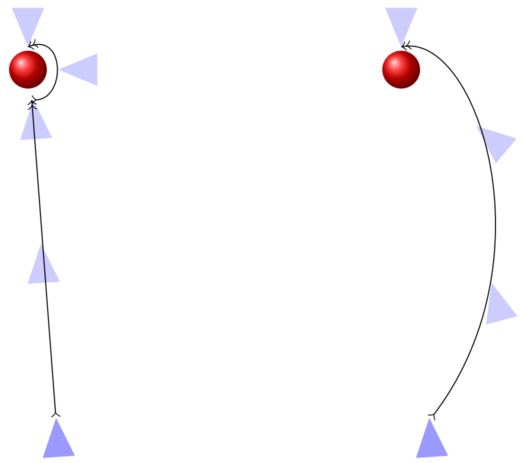
bodySearch Revolve and search for the ball.

bodyStart Initial state when the main code is started up.

bodyStop Stops the robot completely.

bodyUnpenalized Commands the robot to stand back up and walk into the field after being unpenalized.

One important note on BodyFSM is the approaching method. The simplest implementation is to make the robot walk straight to the ball and orbit around it until it faces the attacking goal (ready to make a shot). This method is robust but too slow in a game scenario, where the robots are supposed to approach the ball as fast as possible. In recent years, the curve approach method is introduced, as illustrated in 17(b). Under this implementation, the robot keeps adjusting when approaching the ball. These adjustments are based upon the distance between the ball and the robot, as well as the projected kick direction. As the result, the robot walks in a faster and more natural curve when chasing the ball.



(a) In direct approach the robot spends too much time sidestepping around the ball.

(b) The curve approach allows the robot to perform its angle rotation while walking towards the ball.

Fig. 17. Difference between our original and our improved approach.

5.1.2 BodyFSM for the Goalie

Figure 18 shows the transitions of states for a golaie. It is followed by brief descriptions of the states. The basic routine is: `Track the ball` $\rightarrow$ `Approach the ball if it is near to own goal post` $\rightarrow$ `Kick it away from the gaol post`.

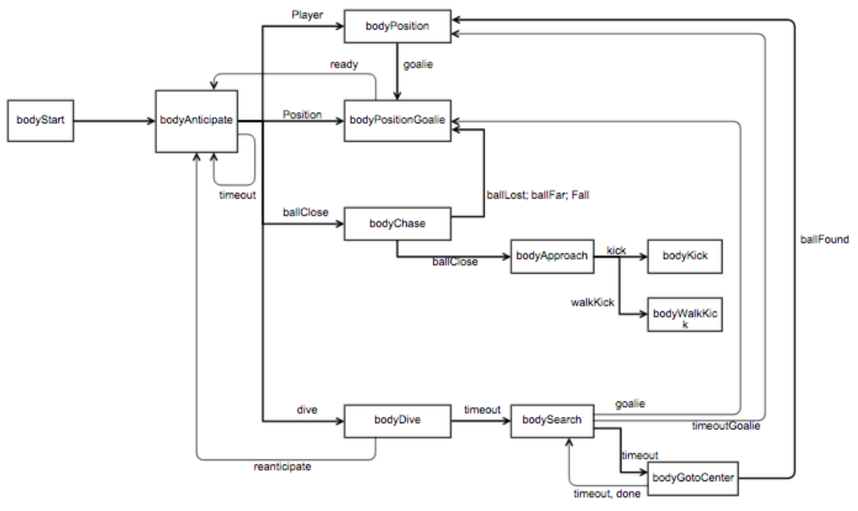


Fig. 18. Body State Machine for a goalie.

bodyStart Initial state when the main code is started up.

bodyAnticipate Predict the near-future position of Goalie.

bodyApproach Align for kick.

bodyGotoCenter Return to the center of the field.

bodyKick Perform stationary kick.

bodyWalkKick Perform a kick while in motion.

bodyPositionGoalie Main body state; most states will transition back here.

bodySearch Revolve and search for the ball.

bodyStop Stops the robot completely.

bodyUnpenalized Commands the robot to stand back up and walk into the field after being unpenalized.

We made some changes recently in goalie behavior. Instead of approaching the ball when ball is less than threshold value from Goalie, the Goalie now approaches a ball if it is near some distance from goal post. This helped Goalie keep highly localized and defend goal post well. Also, Goalie returns to previous position or new best position rather than chasing the ball.

5.2 The Head Finite State Machine

The Head Finite State Machine (HeadFSM) controls the robot head movements. During the game, the robot has to move its head (changes yaw and pitch) efficiently to better and faster locate objects on field. The head movements are usually independent from the body movement, and therefore a separate state machine is designed.

Figure 19 shows the HeadFSM (same for goalie and field players), followed by brief descriptions of the states.

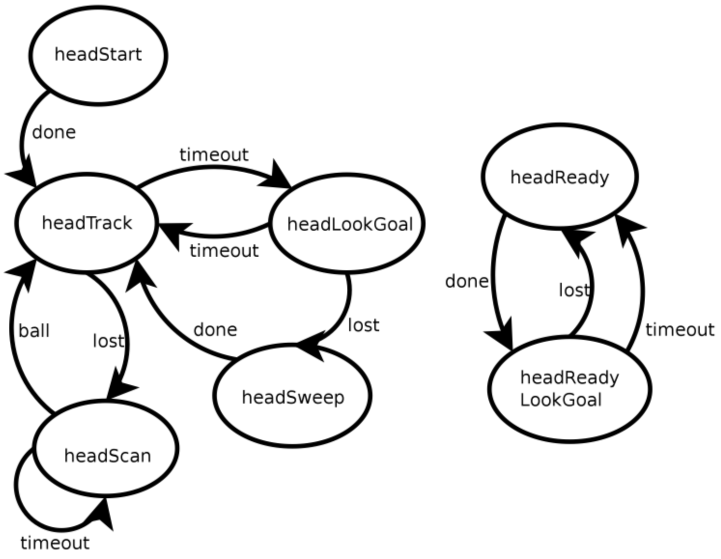


Fig. 19. Head State Machine
Left : during the game / Right: while waiting for game start

headKick During *bodyApproach*, keep the head tilted down towards the ball.

headKickFollow Follow the ball after a kick.

headLookGoal Look up during approach to find the attacking goal posts.

headReady Look for ball while waiting for game start

headReadyLookGoal Look for goal while waiting for game start

headScan Look around for the ball.

headStart Initial state after the game state changes to 'Playing'.

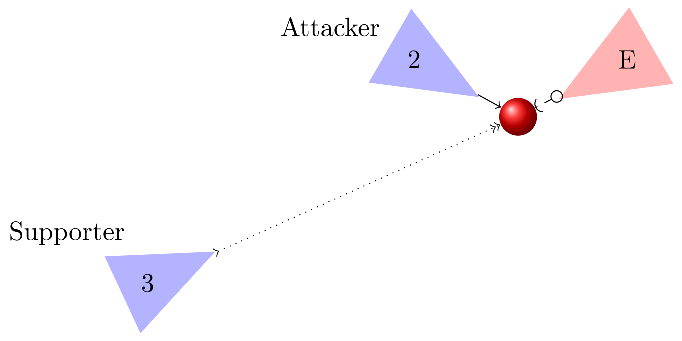
headSweep Perform a general search, with a priority on finding goal posts.

headTrack Track the ball, moving or stationary.

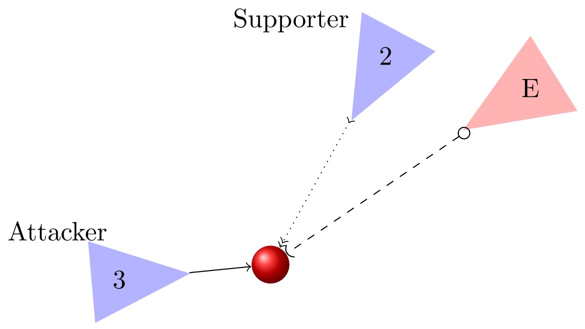
5.3 Team Coordination

Robust single robot behavior is not sufficient to have good performance during robot soccer games. All players on the field have to coordinate and function as a team. Behavior module also regulates team behaviors so robots can make efficient use of the space on field. The infrastructural base of team coordination is the Wifi-based communication between robots, where all players share basic perception information (such as their own locations, the detected ball locations, etc.) The essence of team coordination is a role switching mechanism which helps robots to stay at different locations on the field, contributing to both attacking and defending. There are five defined roles:

Goalie	1	Stays in and around the defensive goal to clear the ball when it comes close.
Attacker	2	Goes directly towards the ball and kicks.
Supporter	3	Follows the attacking robot up-field, but stays at a respectable distance away—usually about midfield.
Defender	4	The defending robot positions itself between the ball and defensive goal area.
Defender Two	5	Performs double duty with the first defender, but has a different initial position.



(a) The ball is closest to Robot 2, the currently Attacker. Robot 3 sights the ball and is assigned Supporter.



(b) The opponent robot kicks the ball and the ball moves toward Robot 3, which becomes the new Attacker. The former Attacker Robot 2 is assigned Supporter due to further distance from the ball.

Fig. 20. Simple Example of Role Switching.

The primary strategy is to keep the ball moving down-field. To encourage this, the four general players (non-goalies) are dynamically switching their roles. According to the shared perception information, each robot calculates its

Estimated Time of Approaching (ETA). ETA is a function of numeral factors, including if the robot is fallen, if the robot sees the ball, the distance between the robot and the ball, walking speed, etc. The robot with the smallest ETA will be assigned the attacker, and other robots will be assigned defender or supporter based on their locations on field. Figure 20(a) and 20(b) is a illustration of dynamic role switching during the game.

Ideally the attacker and support will not fight for the ball. However, this may still happen when the localization is not accurate. In order to avoid that, this year, we started to use ultrasound for obstacle avoidance. The robots will back off if they are running into their teammates, and this also helps prevent robots crushing goalpost.

6 Research Interest and Future Work

While improvements made throughout the years formed the backbone of our strategies to remain competitive in the new surroundings, it is clear that novel techniques will be needed in the near future for reliable play and additional evolutionary rules changes.

Several immediate and short term research focus that is planned to address before RoboCup 2016:

Vision:

- Exploit field context information such as field boundary, to provide more accurate object recognition
- Instead of using current color segmentation based system, implement a peak color detector for field green and Utilize more on edges and geometry features to detect white goalposts and lines so that the robots can play when in natural inconsistent lighting conditions without reliable color features
- Use machine learning techniques for robot detection

Locomotion:

- Rewrite the current locomotion module. Implement closed-loop controller to improve the robot's stability

Behavior:

- Extend the team coordination module to provide advanced techniques for multi-robot cooperation

For long term research goal beyond the scope of competition, we are interested in developing an intelligent system that can adapt and learn from experience. We will focus on learning representations that enable robots to efficiently reason about real-time behaviors and planning in dynamic environments to guide the decision making and reduce uncertainty.

7 Conclusion

The RoboCup SPL is a tremendously exciting community to be a part of. The international competition, aggressive technology development and compelling motivations fostered an environment that brought out a great level of effort from all team members who were involved. Although our software system outlined in this paper is capable of meeting the challenges in RoboCup 2015, there is still much left to do. Future implementations must use more than color cues, as finely trained color classifiers cannot be expected to work in the context of a real outdoor soccer field. Additionally, walking strategies must include models of more realistic field materials in order to remain stable. Team behavior must also meet higher standard for the team to remain competitive.

8 Acknowledgment

The 2015 UPennalizers team gratefully acknowledges the strong and consistent support from the GRASP Laboratory and the School of Engineering and Applied Science at University of Pennsylvania. We also wishes to thank the past team members' contribution to the code base and the organization.