Software Design “State of the Union”
John Alabi, Kent Anderson, Vickie Chiang, Matthew Hill

Top Level State Diagram

Super Happy Funball Game Play

Find Dispenser
- Turn Left
  - Identify beacons

Drive to Dispenser
- Drive Forward
  - (Both on)
  - (Both off)
- Sense tape
  - (Both on)
  - (Both off)
- Turn Left
- Turn Right
  - (Left on & right off)
  - (Right on & left off)
  - (Both on)
  - (Both off, side A)
  - (Both off, side B)

Follow Tape
- Drive Forward
  - (Both on)
  - (Left on & right off)
  - (Right on & left off)
  - (Both on & left off)

Determine playing field orientation (A or B)

Request Balls
- Push Button
  - 1 sec elapsed

Start Scoring

Aim at Target
- Turn Left
- Turn Right

Dispenser Empty
- 2 min Expires
- Stop (hit limit switch)

H

Reach dispenser

Device Driver Level Modules

Beacon Detector Module for both Navigation & Aiming
- InitializeBeaconDetector – Returns nothing, takes nothing. Initializes all necessary hardware and variables for the beacon detector.
- CheckBeaconOnStatus – Returns an unsigned character corresponding to the current beacon detector state. The first two bits correspond to
  - ALL_OFF 0x00
  - RIGHT_ON 0x01
  - LEFT_ON 0x02
  - BOTH_ON 0x03
- CheckBeaconSeenStatus – Returns an unsigned character corresponding to which, if any, beacon is seen.
  - NO_BEACONS
  - GOAL1
  - GOAL2
  - GOAL3
  - DISPENSER

AimingMotor Module
- AIM_TurnRight – takes an unsigned char speed and turns the aimer right
- AIM_TurnLeft – takes an unsigned char speed and turns the aimer left
- AIM_Stop – takes nothing, returns nothing

DriveMotor Module
- SetLeftPWM – takes a signed char DutyCycle and sets that duty cycle for the right drive motor
- SetRightPWM – takes a signed char DutyCycle and sets that duty cycle for the right drive motor

Timer Module
- InitializeTimerModule – initializes a timer at the start of the run that increments with ms precision in a big-endian structure
- GetCurrentTime – Takes nothing, returns the current timer count as a long

TapeSensor Module
- InitializeTapeSensors – takes nothing, returns nothing, initializes sensors
- TSStatus – takes nothing, returns an unsigned char code corresponding to which sensors are active. Example of use:
  ```c
  if((TS_LEFT_ON & TSStatus())
    o TS_CENTER bit 2 hi/lo
    o TS_LEFT bit 0 hi/lo
    o TS_RIGHT bit 1 hi/lo
  ```
Higher Level Modules

Aiming Module
- InitAiming – takes nothing, initializes: turret motor, limit switches, tape measure motor
- CheckAimEvents – returns a code corresponding to an event and runs the event checker for the tape measure
  - L_LIMIT  Detected rising edge on left limit switch
  - R_LIMIT  Detected rising edge on right limit switch
  - NO_EVENT nothing happened

HandleAimEvent

<table>
<thead>
<tr>
<th>Mode:</th>
<th>Hold Left</th>
<th>Hold Right</th>
<th>Turn Left</th>
<th>Turn Right</th>
<th>Hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_LIMIT</td>
<td>OK, very low duty cycle</td>
<td>Turn left, very low duty cycle</td>
<td>Start Hold</td>
<td>Turn right</td>
<td>Nothing</td>
</tr>
<tr>
<td>R_LIMIT</td>
<td>Turn left</td>
<td>OK</td>
<td>Turn left</td>
<td>Start Hold</td>
<td>Nothing</td>
</tr>
<tr>
<td>NO_EVENT</td>
<td>Turn left</td>
<td>Turn right</td>
<td>Turn left</td>
<td>Turn Right</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

*EXTEND and retract maintain the hold command and revert to holding when the action is done.

SetAimMode
Disabled during EXTEND/RETRACT
- TURN_RIGHT
- TURN_LEFT
- HOLD_RIGHT
- HOLD_LEFT
- HOLD
- EXTEND – only if holding, reverts to whatever hold it was doing at the time it was initialized. SetMode is disabled here
- RETRACT – only if holding SetMode is disabled here

GetAimMode

Tape Measure Module
CheckTMEvents –
- MARK_DETECTED
- NO_EVENT

HandleTMEEvent – handles events from CheckTM Events

<table>
<thead>
<tr>
<th>Mode:</th>
<th>Hold</th>
<th>Extended</th>
<th>Retracted</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARK</td>
<td>Nothing</td>
<td>Stop/Hold if t&gt;t_min</td>
<td>Stop/Hold if t&gt;t_min</td>
</tr>
<tr>
<td>NO_EVENT</td>
<td>Nothing</td>
<td>Nothing</td>
<td>Nothing</td>
</tr>
</tbody>
</table>
SetTMMode – sets the tape measure mode to one of the following modes. Note that the modes EXTENDED and RETRACTED should not be permitted to be set external to the module

- EXTENDED
- RETRACTED
- HOLD

GetTMMode – returns the current tape measure mode.

Old Aiming Module – superseded by new version

- InitializingAim – takes nothing, returns nothing, initializes subsystem
- CheckAimEvents – returns a code corresponding to events that may have happened
  - BCN_GOAL1{ _L, _R, _B } Sees goal 1 on Left, Right, Both
  - BCN_GOAL2{ _L, _R, _B } Sees goal 2 on Left, Right, Both
  - BCN_GOAL3{ _L, _R, _B } Sees goal 3 on Left, Right, Both
  - DISP_BEACON Sees dispenser Æ ERROR
  - L_LIMIT Hit left limit switch
  - R_LIMIT Hit right limit switch
  - EXTENDED Deploy Secret Weapon #1
  - RETRACTED Un-deploy Secret Weapon #1
  - NO_EVENT Boring!
- HandleAimEvent – responds to the code from CheckAimingEvents()
- Aim_SetMode – takes a code corresponding to which beacon to look for, returns nothing
  - DISPENSER
  - GOAL1
  - GOAL2
  - GOAL3
  - EXTENDING
  - RETRACTING
  - SHUTDOWN
- IsAimed – Takes nothing, returns TRUE if the aiming subsystem is aiming at the target, FALSE if not.

BeaconNavigating Module

- CheckBNEvents – returns a code corresponding to events that may have happened
  - TRGT_LEFT Target is to left of current heading
  - TRGT_RIGHT Target is to right of current heading
  - TRGT_LOCK Both Detectors register, heading OK
  - TRGT_LOST Target is not visible
  - GOAL1 Sees Goal 1, which is not target
  - GOAL2 Sees Goal 2, which is not target
  - GOAL3 Sees Goal 3, which is not target
• HandleBNEvents – takes event code, responds according to mode
  • BN_Mode
    o GOAL1 Goal 1 is your target
    o GOAL2 Goal 2 is your target
    o GOAL3 Goal 3 is your target
    o DISPENSER Dispenser is your target
    o SHUTDOWN

Driving Module
• VeerRight – Takes uchar speed. Initiates a right pivot about the left wheel.
• VeerLeft – Takes uchar speed. Initiates a right pivot about the left wheel.
• TurnRight – Takes uchar speed. Turn left in place
• TurnLeft – Takes uchar speed. Turn left in place
• Forward - Takes uchar speed. Go straight forward
• Reverse - Takes uchar speed. Go straight back
• Stop – takes nothing, returns nothing, stop all drive motors.

LineFollowing Module
• CheckLFEvents – returns a code corresponding to events that may have happened
  o LEFT_ON
  o RIGHT_ON
  o CENTER_ON
  o FRONT_ON
  o ALL_ON
  o ALL_OFF
• HandleLFEvent – handles the line following event according to the current operating mode
• LF_SetMode – Takes a code corresponding to a mode for line following that determines how the event handler responds
  o TURN_RIGHT
  o TURN_LEFT
  o FOLLOW
  o SEEK_LINE
  o SHUTDOWN

BallRequest Module
• InitializeBR – Returns nothing, takes nothing. Initializes all necessary hardware/variables for the ball requesting functionality
• CheckBREvents – returns a code corresponding to different events
  o BR_READY Ready for a new request
  o BR_PENDING Request is pending (1 s between requests
• HandleBREvents – handles event codes
• BR_SetMode – Sets the BR mode. Modes include
- **SINGLE_BALL**: Request a single ball, resets itself in handler.
- **SHUTDOWN**: Don't Do anything.
- **MAX_BALLS**: Request as many as you can, ASAP.
- **MED_BALLS**: Request continuously at a slower clip.

- **HowManyBallsMade** – returns how many ball requests have been made.
- **Static RequestBall** – Returns unsigned character `NumberOfBalls` that tracks the number of balls, including the current ball, that have been requested. It initiates the requesting of a ball.
- **Static IsRequestFinished()** – continually call this from your event checker after RequestBall. Returns TRUE if done, FALSE if not.
- **Static ButtonReady()** – Returns TRUE if the conditions are met for requesting a new ball from the dispenser. Based on timer count and history of requested balls.
- **Example implementation for max-speed ball requesting**
  ```c
  while(TRUE)
      if(PendingBallRequest == FALSE) {
          if(ButtonReady()) {
              RequestBall();
              PendingBallRequest = TRUE;
          }
      } else {
          if(IsRequestFinished()) {
              PendingBallRequest = FALSE;
          }
      }
  }
  ```