Event Loop & Threads
Finite State Machines
Fault Tolerance
Announcements

- No Food In Lab, for now.
- We're open this weekend.
- Game Board 1 is in lab.
- Large balls are on the way.
- Assignment 3 due Friday.
- Workshop 3 (FlagBox) Thurs, Fri.
- IAP registration closes tomorrow.
Event Loop

- (1) Sense
- (2) Plan
- (3) Act
- Do not proceed to (3) if you haven't done (1) or (2)
Event Loop

Problem:
Lots of computational power, but only a few “reliable” inputs.

Solution:
Use filters, feedback, and failsafe mechanisms.
Sometimes human error is the problem.

AI & Control are non-trivial challenges for engineered systems.

Let's ensure that our code is as correct as possible.
void update_gyro() {
    while (true) {
        //Get angular velocity
        //Adjust for offset
        //Find time difference
        //Angular_dist += angular_velocity * time_difference
    }
}

create_thread(&update_gyro, STACK_DEFAULT, 0, "gyro_thread");
Threads

- Threads communicate through global variables.
- What happens if two threads both want to read/write a variable at same time?
Control Structures

while( !captured_ball() && !crashed_into_wall() ){
    look_for_balls();
}

VS

void infinite_loop(){
    if ( captured_ball() ) return go_to_goal();
    if ( crashed_into_wall() ) return get_free();
    look_for_balls();
}
Naïve State Machine

switch ( state ) {
    case FIND_LINE:
        while ( !detect_line() ){
            drive_forward();
        }
        state = COLLECT_BALLS;
        break;

    case COLLECT_BALLS:
        ...

}
Cleaner FSM

Loop the action until an interrupt is received. Create threads to determine inputs, outputs, and state transitions.

The action code is then virtually independent of its context. Separate threads control when it halts.
• Fault Tolerance
• (video demo)
It is a noisy world!

The world is full of noisy signals.

You need to filter out the garbage.

- What if your bump sensor is lightly tapped at the wrong moment? A wall or ball?

- Your IR picks up an audience member's camera flash.
Redundancy is your friend.

Take multiple reads on your bump sensor (debouncing).

Average your infrared readings over a very small time interval.
Timeouts

You and your opponent chose strategies that collide midway across the map.

How will you detect this?

How do you get back on course?
void Drive( uint8_t motor, int16_t vel, void(*)() Action, uint16_t Delay, uint8_t(*)() StopCondition, uint32_t StopTime) {
    motor_set_vel(motor, vel);
    while((get_time() < StopTime) && !StopCondition()){
        Action();
        pause(Delay);
    }
    motor_set_vel(motor, 0);
}
Use Exit Codes!

#define MISSION_ACOMPLISHED 0
#define WALL_COLLISION 1
#define FOUND_BALL 2

uint8_t drive_forward()
...
return WALL_COLLISION;

return FOUND_BALL;
Other Tips

Use LCD & printf to check values you're testing.

Break a big problem into its components.

Keep your implementations simple at first. Complicate things later.