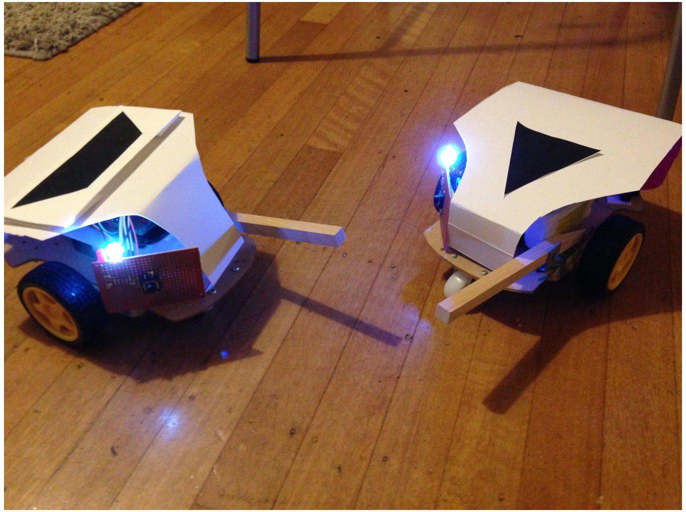
Robot Jousting

A two-player interactive jousting game involving wheeled robots

Alexander Cruz, En Lei, Sunil Srinivasan, Darrel Weng

Project Goal

- Create a physical, interactive ‘jousting’ game using wheeled robots
  - Meet our ‘knights’
  - Player knight (Sir Trap of Zoid)
  - AI knight (Sir Tri of Angle)

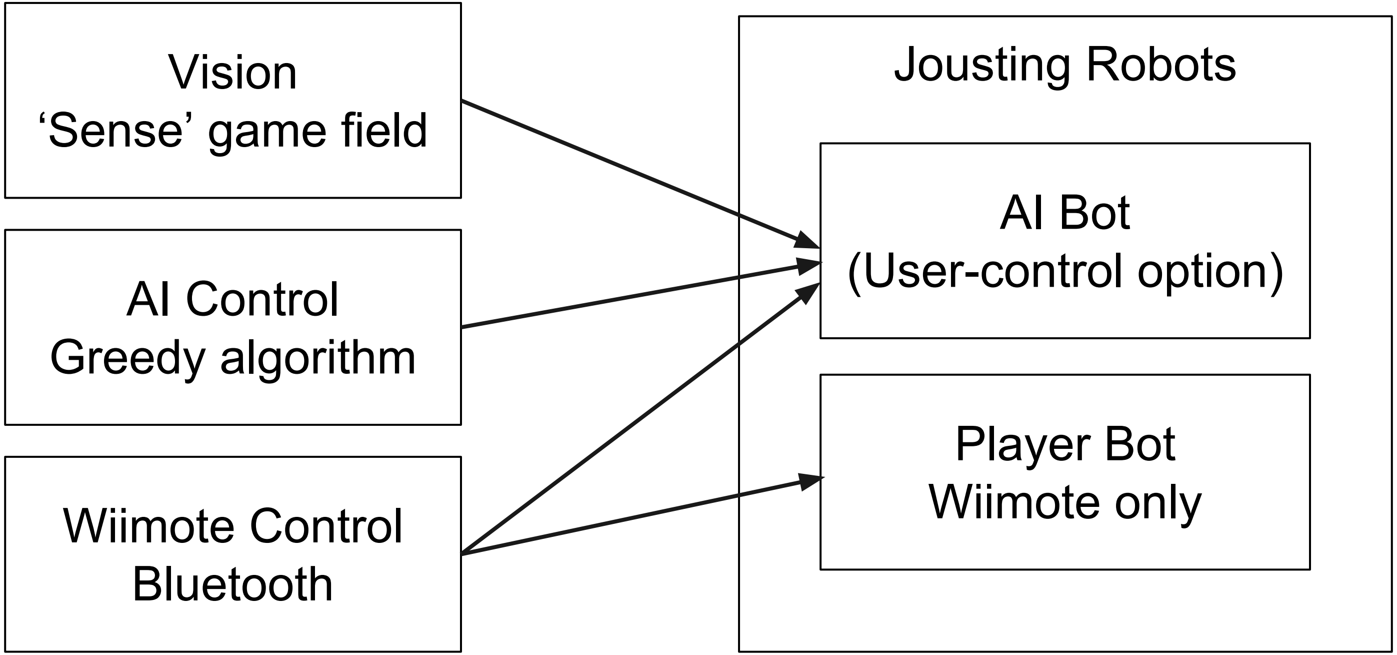


The Game

- Two knights constrained to a Game Field will fight for honor
- Each robot has a hitbox, a joust, and 3 ‘lives’

- Last robot standing wins the game!

Implementation



Vision
‘Sense’ game field

AI Control
Greedy algorithm

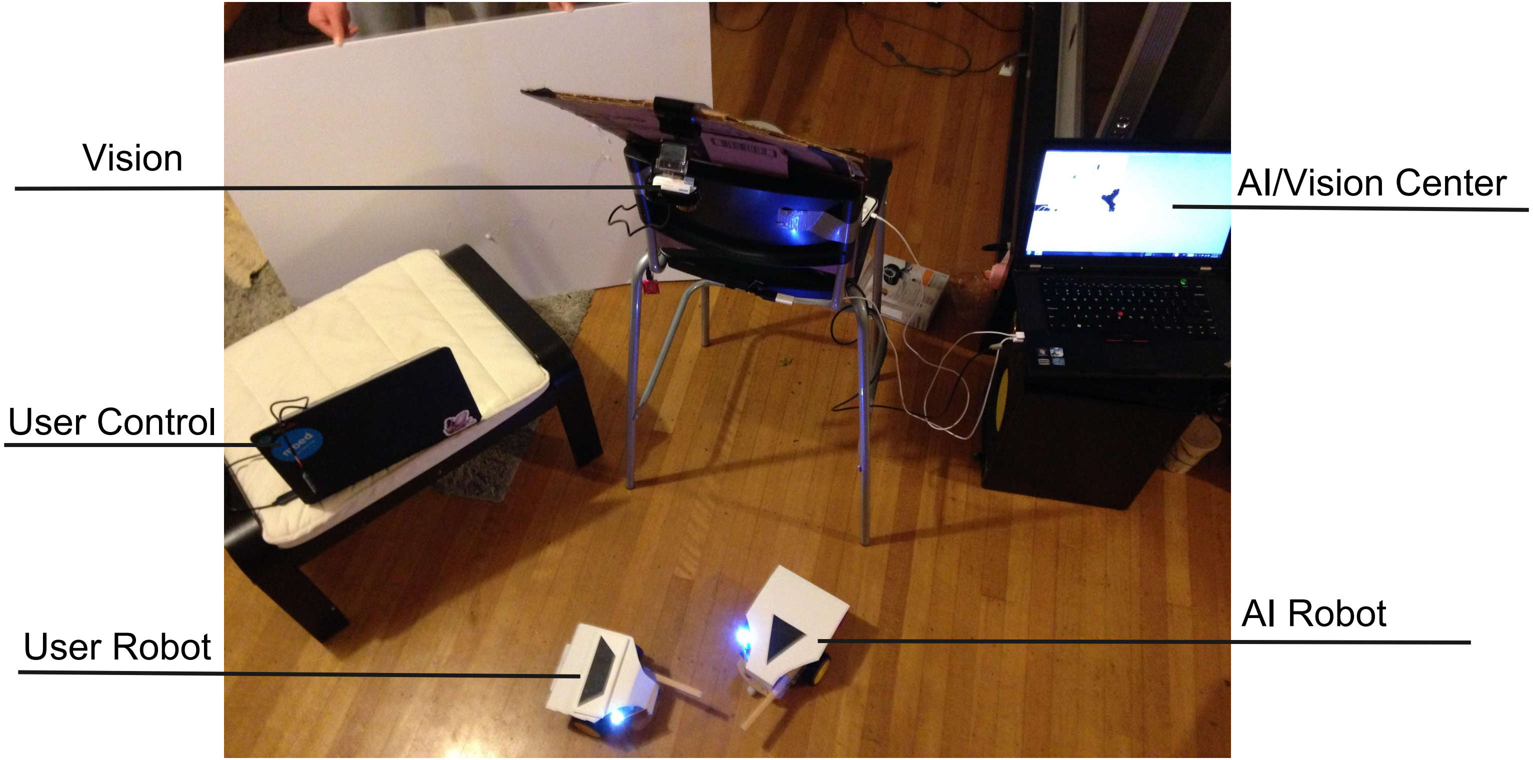
Wiimote Control
Bluetooth

Jousting Robots

AI Bot
(User-control option)

Player Bot
Wiimote only

Our Setup



Part I - Robot Knights

Robots: Design

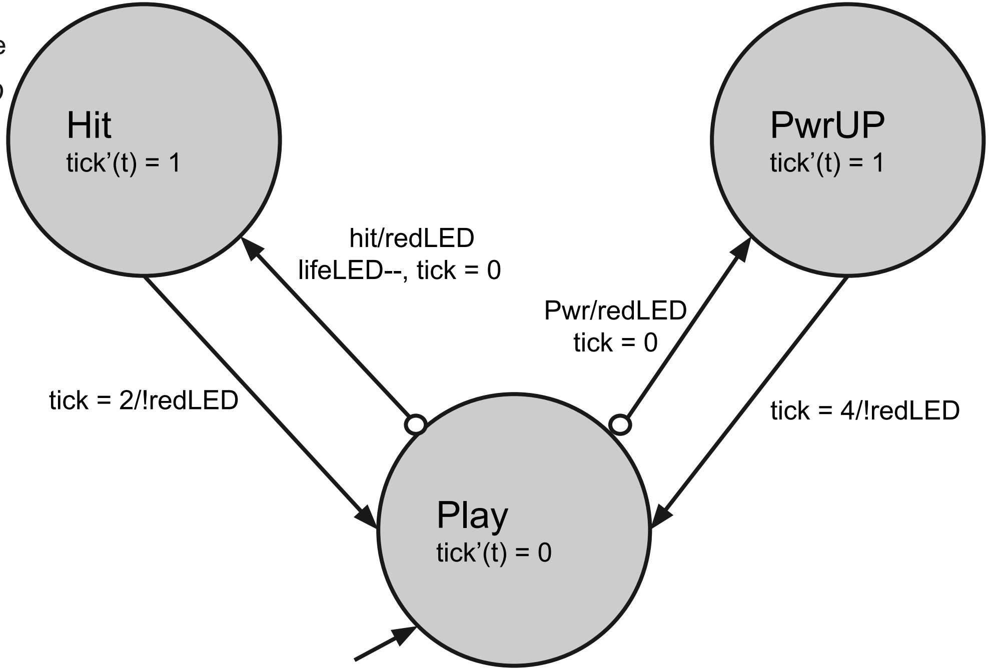
- Main Design:
  - Each robot requires a joust and a hitbox (shield)
- Modeling game mechanics
  - Power-ups: detection & use
    hall sensor + magnets
  - Hit box & Life: implementation and actuation
    push button + RGB LED on a “shield”
- Control & Hardware
  - PWM, serial communication

Robots: State Machine

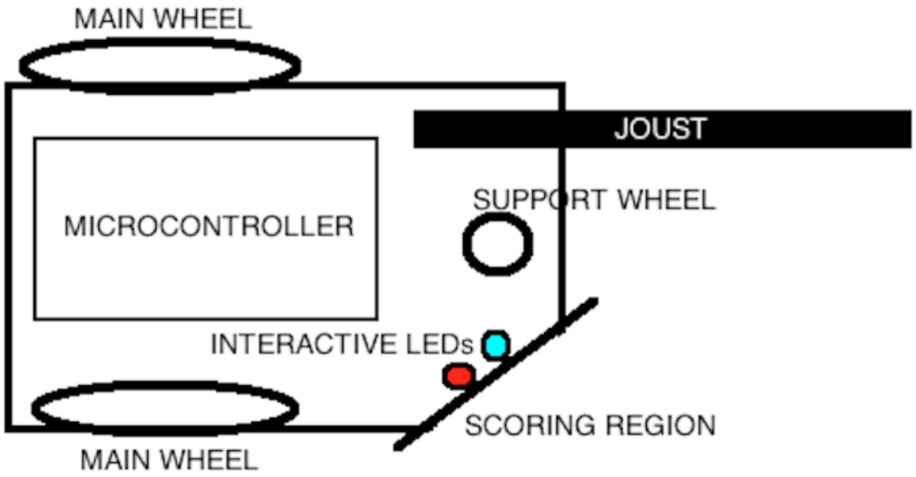
Inputs: *Hit, Pwr*: Pure

Outputs: *redLED*: Pure

Variables: *tick, lifeLED*



Robots: Hardware Layout



- Player Knight:
  - CZ-HC-05 gomcu **Bluetooth** boards
  - PL2303HX USB To TTL To UART Converter
  - FRDM KL25Z **mBed**
  - Radio Shack AA's
- AI Knight:
  - **XBee** Series 1 radio by DigiKey
  - Sparkfun XBee Explorer USB
  - **Arduino** Uno microcontroller
  - Tenergy 7.4V 2200mAh Li-Ion Battery
- Shared Hardware:
  - Pololu DRV8833 Dual Motor Driver Carrier
  - Pololu Adjustable Step-Up/Step-Down Voltage Regulator S7V8A
  - Sunkhee Hall Effect Sensor
  - Motors+chassis from Emgreat Motor Robot Kit
- Different hardware because we wanted to explore using mBed and Bluetooth (vs. Arduino experience)

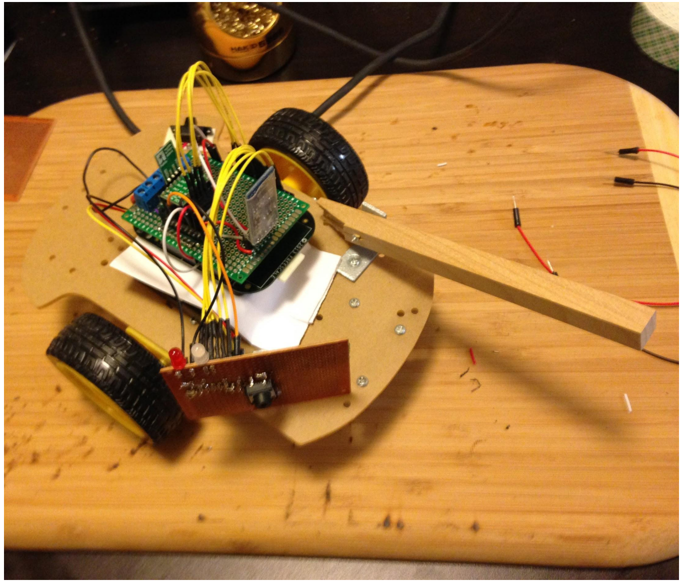
Source: Basic Layout Created using Paintbrush

Robots: Pitfalls

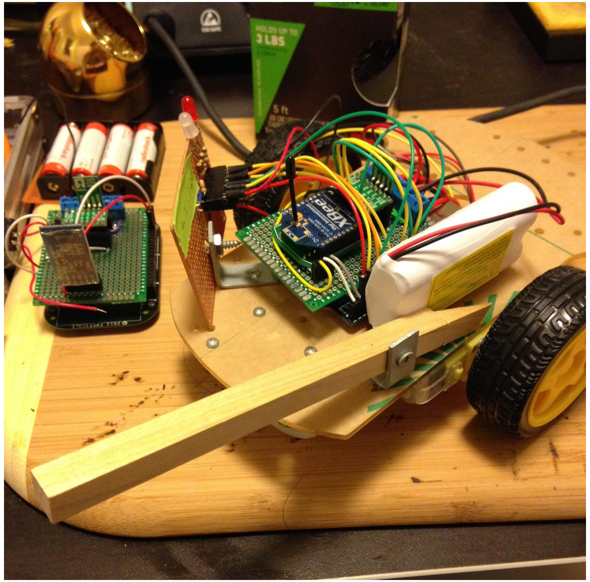
- DC motors from kit produced an unexpectedly huge magnetic field
  - rendered “power-up” mechanic infeasible, as hall sensors would respond to motors
  - created significant interference with HC05 communication (mostly resolved)
- We originally wanted to use a WiiMote, but it turns out that the HC05 is programmed to use only SSP...

Robot Knights

Player Knight



AI Knight



Part II - Vision

Vision: Motivation

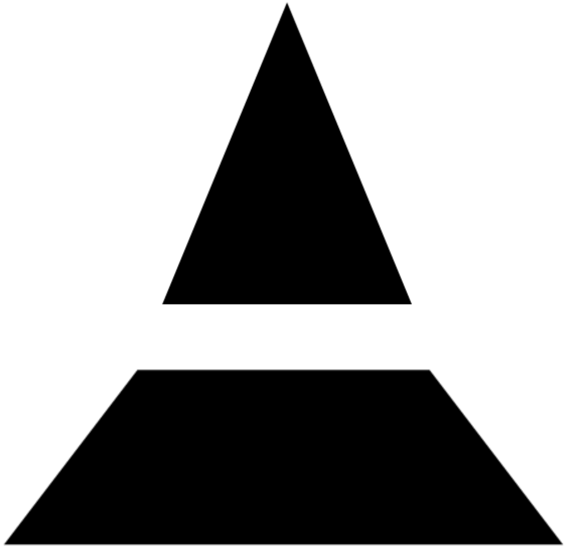
- Our AI knight needs to be able to see!
- Forward-mounted camera doesn’t provide enough information
- Overhead camera as part of the field
- Shape tracking (markers) to determine robot position

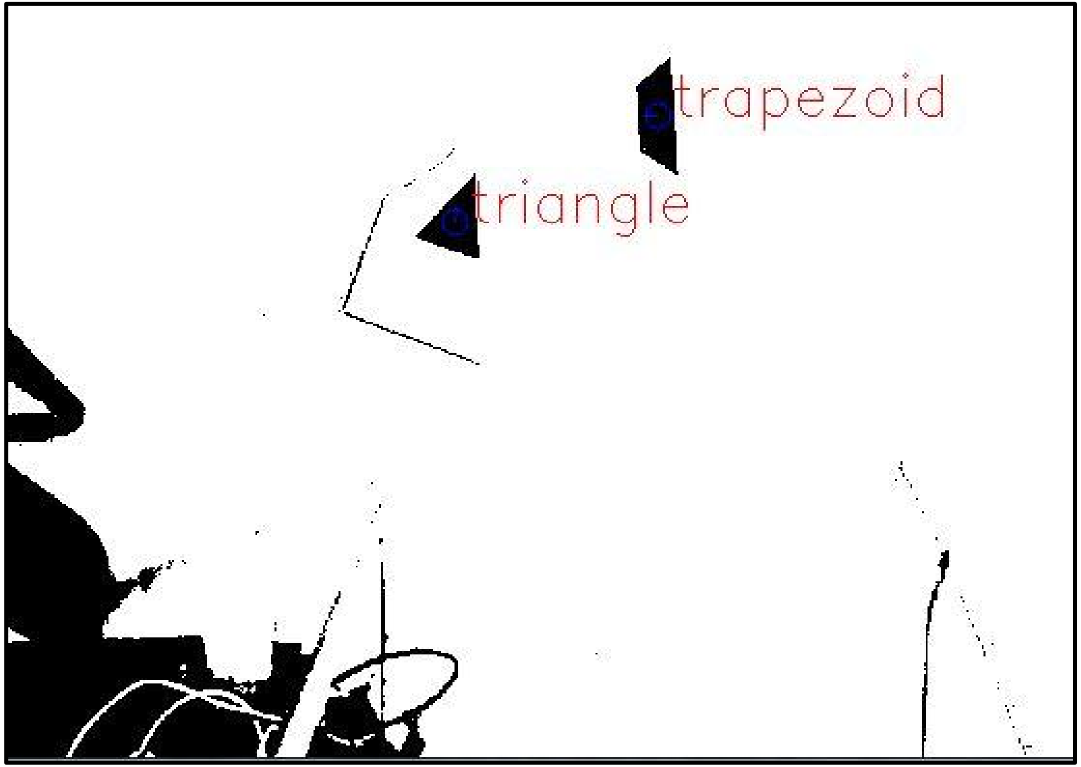
Vision: System

- Image stream from Phillips webcam attached to laptop
- OpenCV Python bindings
  - Get contours and orientations of templates
  - Grayscale image, thresholded to black/white
  - Detect contours in B/W image
  - matchShapes to ‘score’ contours for matches
  - Relative orientation (from template) by angle subtraction modulo 360 (orientation of a contour determined by vector from centroid to center of minimal enclosing circle)
  - Accounts for noise and tiny shape match errors
- “ShapeTracker” class/interface for use by other components
- Here’s an example of what the system sees…

Vision: Example

Templates, B/W image





Source: Images generated by hand, ShapeTracker program

Vision: Issues

- 180° problem
  - Moments - shape descriptors
  - Can only determine orientation of major axis of shape
  - Naive method worked better (centroid to enclosing circle center)
- Contour parents matching incorrectly
  - Discarded any contours with children (only the children were examined) - works for our simple shapes

Part III - AI

AI: Greedy Algorithm

- Our AI uses a greedy/aggressive behavior algorithm that directs the AI knight to actively pursue and try to hit the player knight
- We model our algorithm using a state machine framework coded in Python

AI: Inputs

- The vector orientations of both robots (from vision module)
- The relative angles of the vector orientations of the robots (calculated)
- The distance between the two robots (calculated)

AI: Output

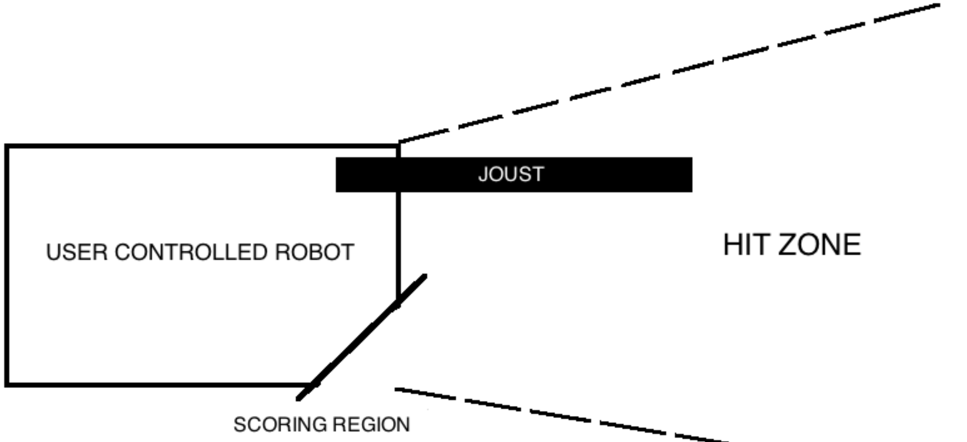
- The next move command for the AI knight
- Possible moves
  - “Go Forward”
  - “Rotate/Turn Left in Place”
  - “Rotate/Turn Right in Place”
  - “Stop”

AI: Evaluation

- Use the given inputs to calculate the relative position and orientation of the player knight with respect to the AI knight
- In general, pursue the player knight
    - E.g. if the user is to the left, AI turns left; if the user is in front, AI goes forward

AI: Evaluation

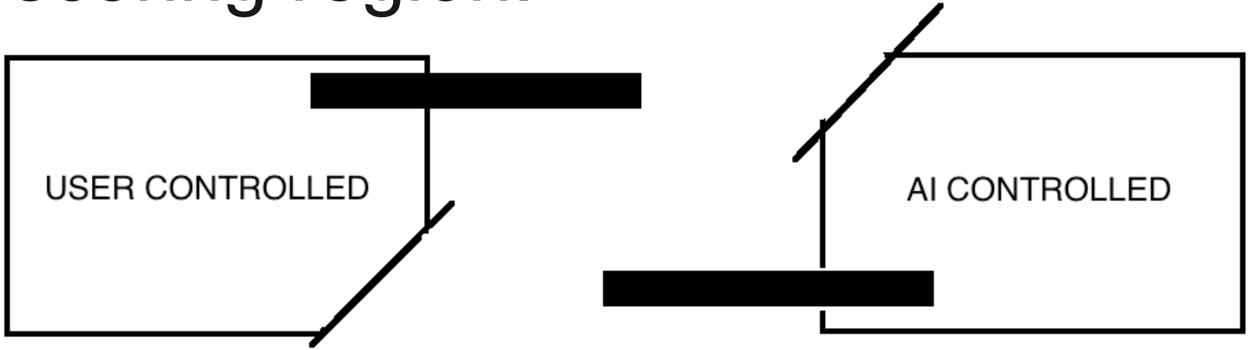
- The one special case is when the AI bot is in the potential hit zone of the user bot



Source: Created using Paintbrush

AI: Evaluation

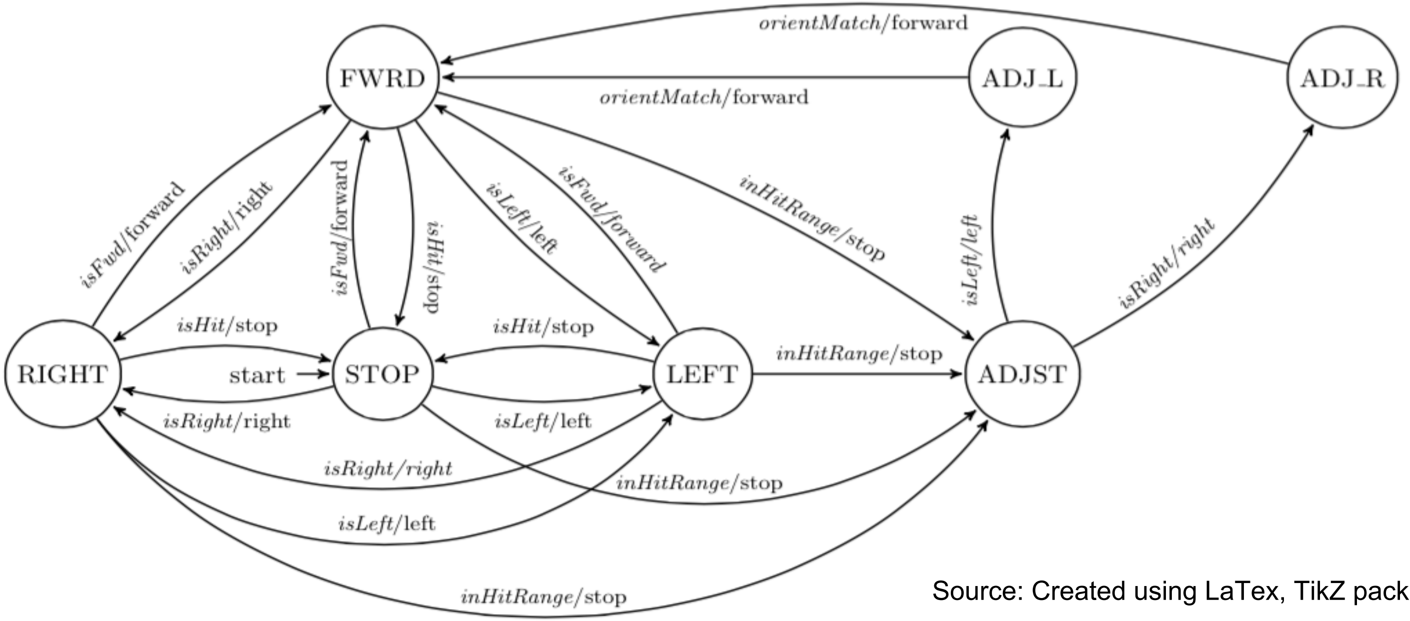
- If the AI finds itself in the hit zone, it adjusts its orientation so its joust is pointing towards the scoring region.



Source: Created using Paintbrush

Inputs: θ_{AI}, θ_{USER}: $\{0,..,359\}$
ϕ_{AI}, ϕ_{USER}: $\{0,..,359\}$
distance: $\mathbb{R}^+$

Output: move: {forward, left, right, stop}



Source: Created using LaTex, TikZ package

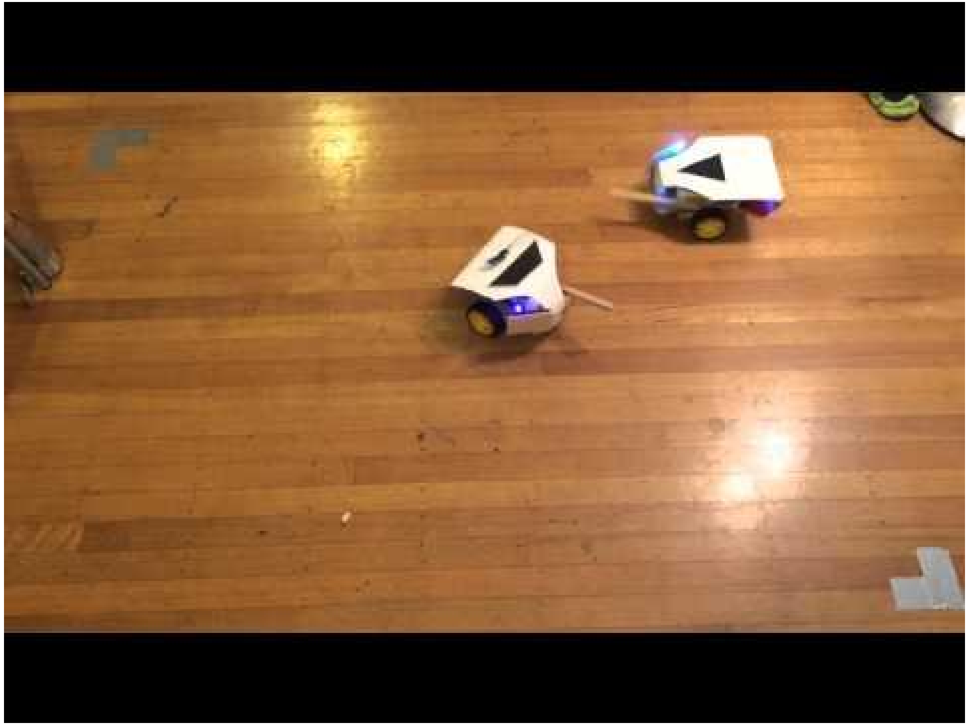
AI: State Machine Diagram

AI: Demonstration

- Determine desired location
  - Path to location
  - Position joust appropriately

Full Demonstration

- We’ll have a demo later today!
- Here’s a video in the meantime
- Try not to snicker



References

- OpenCV
    - http://opencv.org/
- Python
    - http://python.org/
- Anaconda by Continuum Analytics
    - http://store.continuum.io/cshop/anaconda/
    - Python distribution, NumPy
- ARM mBed
    - http://developer.mbed.org/
- Arduino
    - http://arduino.cc/

Questions/Comments?