Machine Learning for Controlled Slides of a RC Car
CS229 Term Project

Morgan Quigley and Michael Vitus

December 16, 2005

1 Hardware

The hardware platform for this project was basically unstoppable.

1.1 Drivetrain

For the drivetrain and mechanical platform of our car, we started with an XRay M18 kit and added a 150-watt brushless motor, its matching 3-phase power converter, and a 3-cell lithium polymer battery. We found that this drivetrain produces enormous amounts of power, far beyond what we truly needed. But power is addictive and you can never have enough of it. To allow continous drifting of the car, we covered the tires with small pieces of 1.25" PVC pipe. The completed drivetrain is shown in Figure 1.

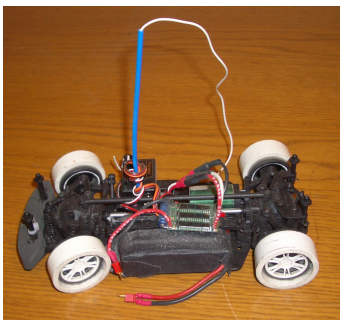


Figure 1: Brushless (3-phase) drivetrain with LiPo battery and 72 MHz radio receiver

1.2 Control

Using hardware designed by Mark Woodward, our software was able to generate standard 72 MHz radio control signals. Mark fabricated an Atmel-based

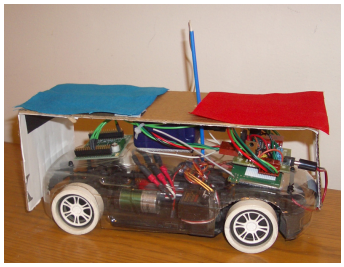


Figure 2: Final assembly with vision markers installed

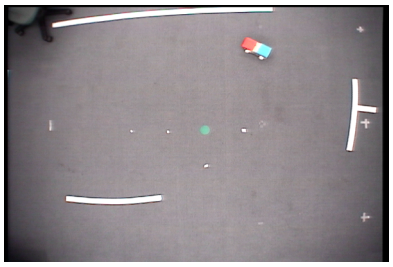
board which accepts R/C stick positions via RS-232 and outputs a standard R/C PWM pulse train which is fed into the trainer port of an R/C transmitter. The transmitter's trainer switch is then used to choose between autonomous and manual control of the car. A standard R/C radio receiver was used on the car to control the steering servo and provide throttle input to the motor's speed control.

1.3 Localization

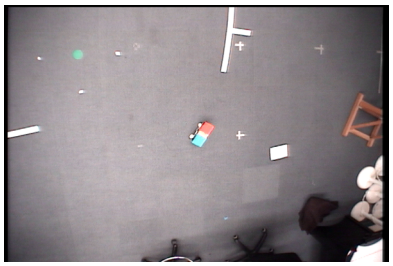
As shown in Figure 2, we attached squares of red and blue fabric to the front and rear of the car so that we would be able to derive the position and heading of the car from an overhead camera. The overhead camera is able to pan and tilt to see a wide area of floor space, but we found that when the camera was in motion, latency from its pan and tilt encoders threw the position estimates far out of whack. To reduce this effect, we discretized the camera's motion into 3 settings of pan and tilt (for a total of 9 possible camera attitudes).

In addition, we added a green dot to the middle of the playing field, and differenced the car's computed position from the computed position of the green dot. We found that this dramatically reduced the errors from the moving camera, as the encoding errors will

equally skew the positions of the green dot and the car, leaving their **relative** positions intact. We chose the camera pan and tilt discretizations so as to ensure that the green dot at the origin never left the camera frame. Two typical camera frames are shown in Figure 3.



(a) Camera centered



(b) Camera panned to a corner

Figure 3: Two typical camera frames. The green dot on the floor marks the center of the coordinate system. The car's red and blue squares give both position and heading. (The white tape on the floor is not related to our project.)

After some experimentation, we were able to tweak Mark's vision code into running at 30 frames/sec, which is the full frame rate of the camera.

1.4 Onboard Inertial Sensing

We soon found that the yaw rate estimates from the vision system were noisier than we wanted. This was unsurprising, as the yaw estimate itself is computed as the arctangent of the coordinate distances between the front and rear markers on the car. Taking the numerical derivative of this already-differenced yaw estimate introduced significant noise into the system. To overcome this, we added a black-box IMU (an XSens MT9) to the car. Among the many sensors in the IMU is a z-axis MEMS rate gyro, which provided a direct measurement of yaw rate. We attached a 900 Mhz radio transceiver to the car, and used a Rabbit microcontroller to parse the data streaming out of

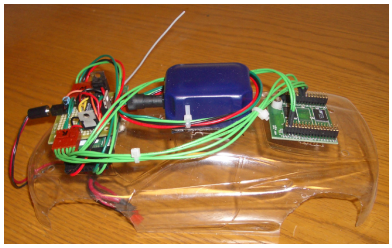


Figure 4: Interface board stacked on 900 Mhz transceiver at left, IMU at center, and Rabbit microcontroller at right

the IMU, pick out the z-gyro readings, and transmit them 30 times per second over the 900 MHz link. We fabricated a board which regulated the car's LiPo battery to 3.3V, 5V, and 6V as needed by the various parts of the system, and performed level conversion between RS-232 (used by the IMU) and 3.3V TTL (used by the microcontroller). These modules were attached to the roof of the car, as shown in Figure 4.

A 900 MHz transceiver was connected to the controller PC to receive the gyro data. To calibrate and unbias the gyroscope measurements, we drove the car for several minutes, and performed linear regression of the vision yawrate estimate against the z-gyro yawrate measurements. A typical plot of this data is shown in Figure 5. Running the gyroscope readings through this linear map resulted in the desired scaling of radians/sec and zeroed the sensor's dc bias.

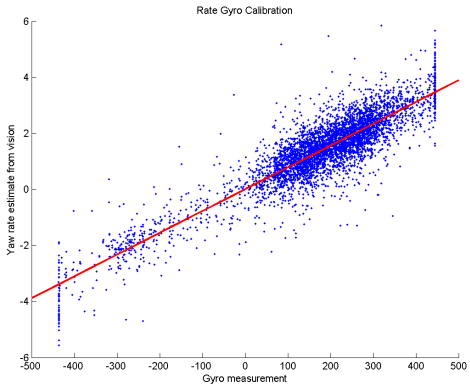


Figure 5: Calibration of the yaw rate gyro measurements to radians/sec. Sensor saturation can be seen at extreme right and left, but this was outside the normal operating range of the experiment.

2 Model

After the hardware was assembled, the next step of the project was to develop a model of the car during controlled slides. A body frame was used in modeling of the car to take advantage of the symmetries of the system which is shown in Figure 6. In normal operation of a car, the angle between the velocity vector and the heading of the car, β, is in the range of $\pm 8^o$, and therefore it was used to distinguish when the car was sliding.

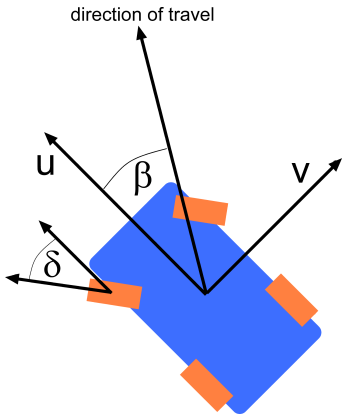


Figure 6: Coordinate frame of the car.

The states of the car that are the most important to model at each time step are the forward velocity of the car, u, the cross-track velocity, v, and the rate of rotation around the vertical axis, r.

A discrete time model instead of a continuous time model was used to model the car's dynamics because the data collected of the car was limited by the frame rate of the camera which is 30 Hz. The model of the car's dynamics is shown in Equation (1) in which a higher dimensional mapping, Ψ, is used for the current state and current input to predict the state at the next timestep.

$$x\left(t+1\right) = A\Psi^{x}\left(x\left(t\right)\right) + B\Psi^{u}\left(u\left(t\right)\right) \qquad (1)$$

where $x\left(t\right) = \left[u\ v\ r\right]^{T}$, $u\left(t\right) = \left[\delta\ \tau\right]^{T}$, δ is the steering input, and τ is the throttle input. To determine the model parameters, A and B, least squares was used to minimize the error between the estimate of the state and the actual state observed, which is shown in Eqn. (2).

$$\min \sum_{t} \left\| x\left(t+1\right) - \left[A\Psi_{x}\left(x\left(t\right)\right) + B\Psi_{u}\left(u\left(t\right)\right)\right] \right\|^{2} \qquad (2)$$

Numerous feature mappings were explored in determining the optimal trade-off between variance and bias of the training data. The final feature mappings that were used in determining the model were $\Psi_1^x\left(x\left(t\right)\right) = \left[u\ v\ r\ ur\ vr\ u^2\ v^2\ r^2\ u^3\ v^3\ r^3\right]^T$ and $\Psi_1^u\left(u\left(t\right)\right) = \left[1\ \delta\ \tau\ \delta^2\ \tau^2\ \delta^3\ \tau^3\ \delta\tau\right]^T$.

To evaluate the validity of the model, the data that was parsed from the driving logs was integrated forward in time to estimate the position of the car. The average of the error for each time step was plotted versus time for two different feature mappings in Figure 7 where $\Psi_2^x\left(x\left(t\right)\right) = x\left(t\right)$ and $\Psi_2^u\left(u\left(t\right)\right) = u\left(t\right)$.

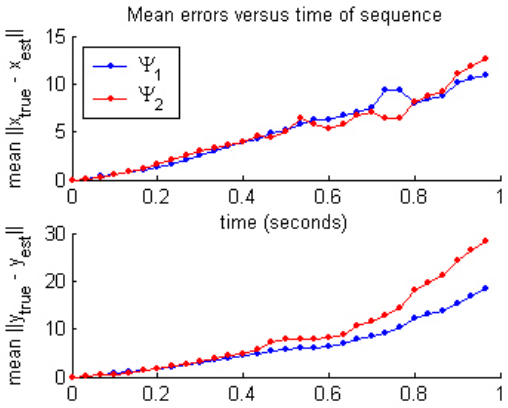


Figure 7: Mean errors versus time of sequence.

The higher dimensional feature mapping, Ψ_1, provides slight better integration error than using just the current state and input to predict the next state. To obtain a feel for the coefficients, for the sake of space, the following A and B matrices are the coefficients for the lower dimensional feature space, Ψ_2.

$$A = \begin{bmatrix} 0.97 & 0.016 & -1.55 \\ 1.21 \times 10^{-4} & 0.95 & -2.5 \\ 0 & 5 \times 10^{-4} & 0.99 \end{bmatrix}$$

$$B = \begin{bmatrix} -9.61 & 58.13 \\ -0.06 & -6.36 \\ -0.08 & 0.30 \end{bmatrix}$$

With a valid model of the sliding dynamics, the next step was to learn how to control the coordinated slides.

3 Controller

We experimented initially with an LQR controller, but decided to implement a Q-learning controller.

Using the model, Ψ_1, produced in the previous section, the Q-learner "drove" the simulation thousands of times. The desired trajectory of the car is a coordinated slide of radius 1m which is shown in Figure 8.

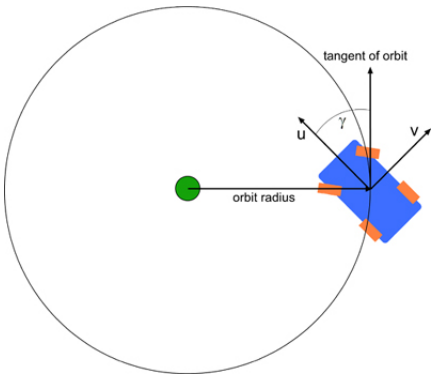


Figure 8: Desired trajectory of controlled slide and car states.

The states that were used to discretize the state space are $(r, \dot{r}, \gamma, \dot{\gamma})$. The reasoning behind the choice of these states is that for the desired trajectory they should remain constant, and therefore, it is linearizing the system around an operating point.

The ranges of the discretization are:

$$\begin{array}{ll} r < 20\ cm & 20\ cm \leq r < 50\ cm \\ 50\ cm \leq r < 90\ cm & 90\ cm \leq r < 110\ cm \\ 110\ cm \leq r < 150\ cm & 150\ cm \leq r < 190\ cm \\ r \geq 190\ cm & \end{array}$$

$$\begin{array}{ll} \dot{r} < -100\ \frac{cm}{s} & -100\ \frac{cm}{s} \leq \dot{r} < -30\ \frac{cm}{s} \\ -30\ \frac{cm}{s} \leq \dot{r} < 30\ \frac{cm}{s} & 30\ \frac{cm}{s} \leq \dot{r} < 100\ \frac{cm}{s} \\ \dot{r} \geq 100\ \frac{cm}{s} & \end{array}$$

$$\begin{array}{ll} \gamma > 2\ rad & 2\ rad \geq \gamma > 1.25\ rad \\ 1.25\ rad \geq \gamma > 0.5\ rad & 0.5\ rad \geq \gamma > -0.25\ rad \\ \gamma \leq -0.25\ rad & \end{array}$$

$$\begin{array}{ll} \dot{\gamma} < -2\ \frac{rad}{s} & -2\ \frac{rad}{s} \leq \dot{\gamma} < -1\ \frac{rad}{s} \\ -1\ \frac{rad}{s} \leq \dot{\gamma} < 0\ \frac{rad}{s} & 0\ \frac{rad}{s} \leq \dot{\gamma} < 1\ \frac{rad}{s} \\ 1\ \frac{rad}{s} \leq \dot{\gamma} < 2\ \frac{rad}{s} & \dot{\gamma} \geq 2\ \frac{rad}{s} \end{array}$$

The steering actions that the controller was able to choose between are: steer straight, left and hard left, and the throttle actions are: slow, medium and fast. Since the dynamics of the car during sliding are unstable, erratic manuevers, such as constant switching of steering angle, will cause the system to become unstable and enter into "donuts". Therefore, for the reward function, we penalize the controller for changing the control action. At each timestep, the reward structure was as follows:

$$R = R_o - 2\left(|\delta_{new} - \delta_{old}| + |\tau_{new} - \tau_{old}|\right) \quad (3)$$

where

$$\begin{array}{lcll} R_o & = & +1 & if\ \ 90\ cm \leq r < 110\ cm \wedge \\ & & & -30\frac{cm}{s} \leq \dot{r} < 30\ \frac{cm}{s} \wedge \\ & & & -1\ \frac{rad}{s} \leq \dot{\gamma} < 1\ \frac{rad}{s} \\ R_o & = & -1 & if\ \ r < 30\ cm \vee r > 190\ cm \\ R_o & = & 0 & otherwise \end{array}$$

Whenever R_o was -1, the simulation was restarted, just as in the pendulum problem in the problem set. The update step for the Q-function is:

$$Q(s,a) = (1-\alpha)\,Q(s,a) + \alpha\left(R + \gamma \max Q(s',a')\right) \quad (4)$$

An example of the simulation run after the controller has trained the Q-function is shown in Figure 9.

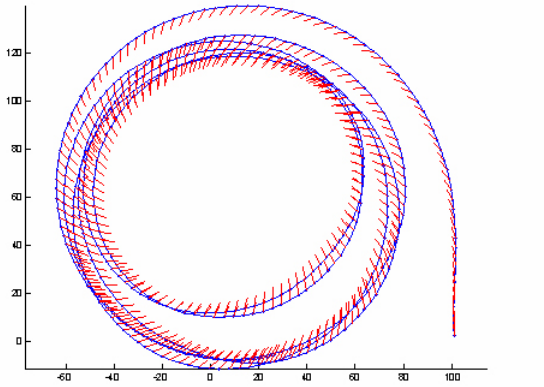


Figure 9: Simulation of the model and controller.

4 Real-World Validation

To drive the real system, we simply exported the Q-table from MATLAB, copied it to the PC hooked up to the vision and control systems, and executed the policy encoded in the Q-table in realtime. Generating the controller's state $(r, \dot{r}, \gamma, \dot{\gamma})$ in realtime from the output of vision system and the IMU took significant debugging efforts. However, eventually we were able

to compute this information, discretize it, perform the Q-table lookup, and send out the steering and throttle commands in realtime.

It would be an overstatement to say that we were entirely pleased with the results. We found that the "handoff" of control from the human driver to the computer was tricky to get right. Because our model only learned how to drive the car when it was sliding, the human driver needed to initiate a slide and then give control to the computer. In addition, the Q-learner only learned how to control the car within a certain range of initial conditions of $(r, \dot{r}, \gamma, \dot{\gamma})$. Although we randomized these initial conditions during training, it still took some practice and patience to execute the control "handoff" when the car was in a feasible state.

Camera motion was also a difficulty; although the green dot at the origin of the coordinate system helped dampen spurious measurements during camera motion, there was still some residual error in the system during camera slew, and this often caused the state estimates to be incorrect. We also suspect that the overall system latency, including vision processing, action selection, control waveform synthesis, and actuator response, lead to the imprecise results. However, even with all of these issues, on a good run, the Q-learner ended up controlling the car better than we were able to do manually. Figure 10 shows the best orbit the computer was able to perform, along with the ideal 1-meter orbit.

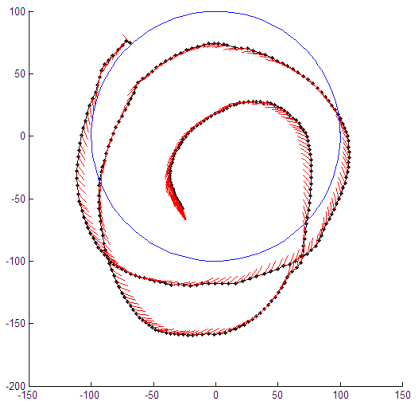


Figure 10: Best performance from the controller with the real-world system. The car entered the orbit at upper-left, and did 1.5 circles before sliding out of control in the center of the orbit.

During the trial shown in Figure 10, the car's sideslip angle ranged from 5 and 50 degrees. The Q-learned policy was able to control this sideslip angle to maintain some semblance of an orbit. A plot of the sideslip angle experienced during the trial is shown in Figure 11.

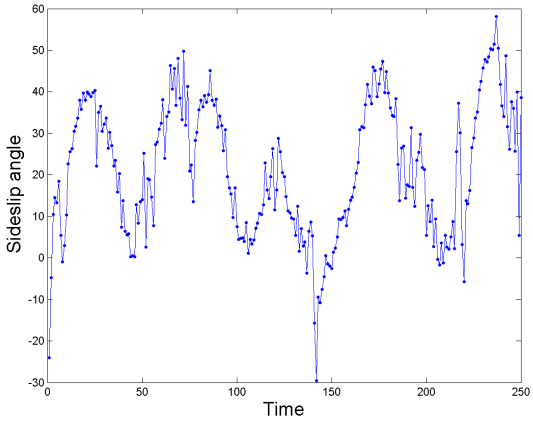


Figure 11: Sideslip angle during the trial.

5 Acknowledgments

We must acknowledge the help and advice of Mark Woodward, who designed and built the R/C transmitter interface hardware and software, and constructed the vision system. Without his help, this project would have never happened in a single quarter. We also acknowledge Pieter Abbeel and Adam Coates in their advice for the Q-learner. In addition, Kurt Miller helped us understand the IMU and 900 MHz radios.