Autonomous Parallel Parking System

Ayush Shah, Vidhi Raval

Abstract— Parallel parking a car is a very challenging task for every driver. A major problem while parallel parking is that the available space is not used in an optimum manner. Also there are high chances of accidents occurring. Human judgment and perception makes it difficult to parallel park cars. In this paper, we discuss in detail a simple and cost effective yet accurate autonomous parallel parking system. External infrared sensors, a parking space detection algorithm and finally, a parking algorithm, enable the process of autonomous parallel parking.

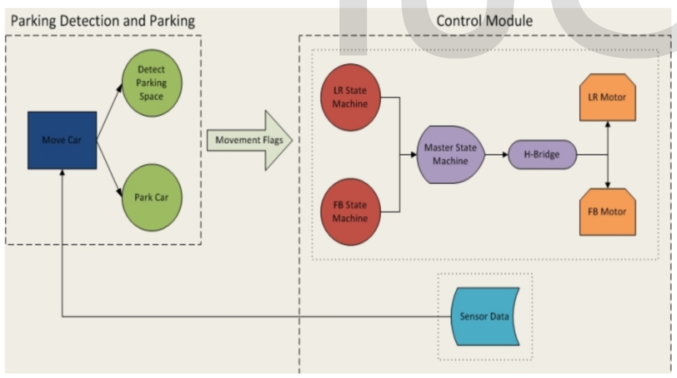
Keywords— Autonomous parking system, Parallel parking, Parking sensors, Detect space, Move car, Park car, Space optimization

1. Introduction

Our aim is to design a car that drives down a street searching for a parking space to its left using distance sensors. When the car has identified a space, it checks to see whether that space is small enough to park itself in the available space. If it determines that there is sufficient space, the car will begin parallel parking into that space. It uses information from sensors placed on the front, left and rear of the car to direct the car into the parking space. Once the car has parked itself, it will remain in that position until it is reset.

2. Logic Structure

The software is broken down into two major components: the control system algorithm and the move car algorithm. The move car algorithm directs the car and the control system implements the directions of the move car algorithm.



2.1 CONTROL SYSTEM ALGORITHM

The control system contains all the hardware and its associated software. It allows the parking and parking detection algorithms to interface with the car. The software in this module is broken up into three major sections: the Left-Right/Front-Back (LR/FB) state machines, master state machine, and distance calculations. The LR/FB state machines determines which direction to move the car based on flags set by the detect parking space and park car algorithms. Once the LR/FB state machines decides which direction to move the car, the master state machine implements this movement by sending the correct input and enable signals to the H-Bridge. The distance calculations implemented independently every millisecond.

The state machines in ControlModule control the motors, and are:

- fbStateMachine
- lrStateMachine
- masterStateMachine

a) fbStateMachine()

Function: The fbStateMachine controls the motor for Forward-Backward operations. It is controlled by the isForward and isReverse flags. These flags serve as indicators as to whether the car should be traveling forward or reverse. In order to control the velocity of the forward-backward motion we anded the enable bit with a PWM signal.

Working: In State 0, the motor is at rest. The corresponding FB control bits are 00. When the algorithm requires the car to go forward or reverse, the corresponding flags (isForward and isReverse) are set, and the FB state machine switches states to 1 or 3 respectively.

In State 1, the motor rotates to drive the car forward. The state machine remains in this state while isForward is equal to 1. Once isForward is de-asserted, the state machine moves to a buffer state to stop the car from moving forward due to inertia. After isForward is set to 0, leaving state 1 and stopping the motor isn't enough. The wheels might continue to rotate due to inertia, and so a buffer state, State 2, is required. It makes the motor go in Reverse for 1 cycle (50ms) of the FB State Machine, before going back to the rest state, State 0. If isReverse is asserted, the state machine jumps to State 3. The state machine remains in this state while isReverse is equal to 1. Once isReverse is de-asserted, the state machine moves to a buffer state to stop the car from moving in reverse due to inertia.

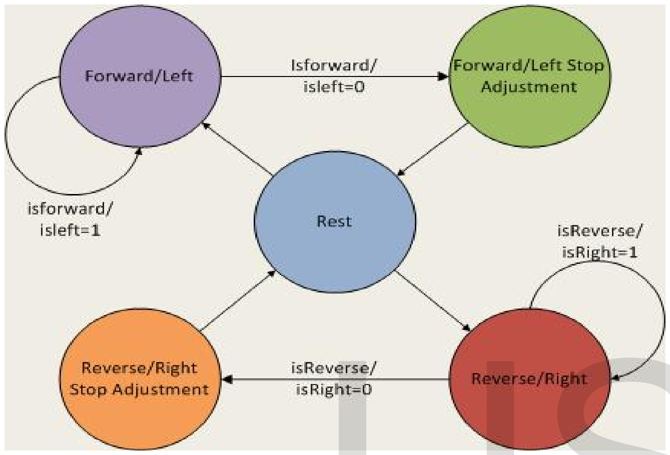
After State 3, a buffer state, State 4, is needed to stop the wheels from continuing to rotate in reverse due to inertia. This is a 1 cycle Forward motion, similar in function to State

2's reverse functionality. Once done, the FB State Machine goes back to its rest state, State 0.

Timing: The fbStateMachine is called upon every 50ms. This is enough time to evaluate the flags set in the AlgorithmModule, but at the same time fast enough to make the motor motion very accurate.

b) lrStateMachine()

The lrStateMachine() works the same way are the fbStatemachine. A forward corresponds to a left turn and a right corresponds to a reverse. The diagram for both are:



c) masterStateMachine()

Function: This uses the FB and LR control bits to call the required functions in order to send the appropriate input signals to the H-Bridge and make the motors rotate in the appropriate direction.

Working: In this function, the 2 FB and LR control bits are combined to create 4 master control bits by left shifting the FW bits by 2 and adding it to the LR bits.

Therefore,
fbBits = fb.controlBits; // (FB FB)
lrBits = lr.controlBits; // (LR LR)
masterBits = (fbBits<<2) + (lrBits); // (FB FB)(LR LR)

As a result, each of the 7 car movements (stop, forward, forward-left, forward-right, reverse, reverse-left, reverse-right) have a unique masterBits combination associated with them. The master control bits are then used in the function to decide which motor control function is to be called.

Timing: This state machine is invoked in each iteration of the infinite while loop in main. In other words, it can be considered to be executing continuously and not at intervals. This is essential because parking requires a great deal of accuracy when controlling the motors. Therefore, we want to update the motors as often as possible, which would require us to call masterStateMachine as often as possible.

2.2 Move Car Algorithm

Move car contains the detect parking space and parallel parking algorithms. All functions in move car interface with the control module by setting movement flags. The parking space detection and parking algorithms use information from the distance sensors to set these movement flags and guide the car. Move car works by initializing the movement flags of the car. It sets the car on a default trajectory and then calls detect parking space. Once a parking space has been detected, the parking algorithm is called. After the car has successfully parked, it idles until it is reset.

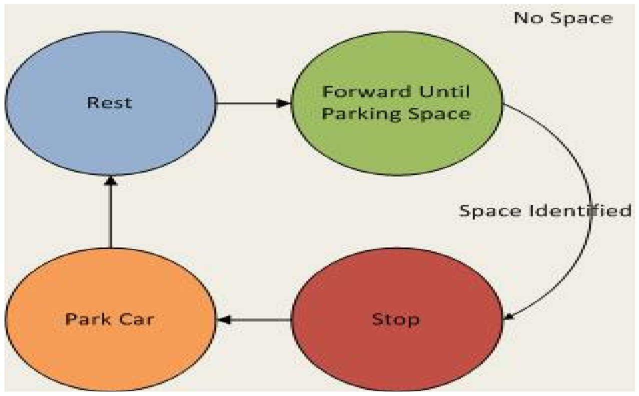
The state machines in the AlgorithmModule use the sensor data and various algorithms to determine what should be the next movement the car must make. They assert flags which tell the ControlModule state machines to actually move the motors. The state machines in AlgorithmModule are:

- moveCar
- detectParking
- parkCar

d) moveCar()

Function: This is the master state machine of the algorithm module. It decides which mode the car is in, i.e., whether the car is moving forward to detect a parking spot, aligning itself once a parking spot has been detected, or actually going through the motion of parking.

Diagram:



Working: This is a 5 state linear state machine, as can be seen in the diagram above.

It starts off in State 0. In this state, the car is at rest. It gives enough time for all transients in the car to stabilize. Once everything is stable, it moves to State 1. In State 1, car moves forward till it detects a parking spot. While in this state, the car invokes the detectParking state machine each time the moveCar state machine is called in the Control Module. Details of how the detectParking state machine works are explained in the next section.

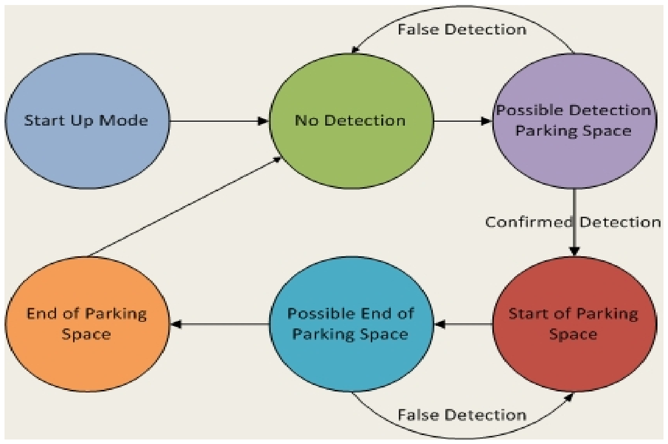
Once a parking lot has been detected, the state machine moves into State 2. It remains in State 2 until the car has parked itself. The parkCar state machine is invoked for each cycle that the moveCar state machine is in State 2. Once the car has been parked by parkCar state machine, the isParked flag is asserted, and moveCar moves onto state 3. When we reach State 3, the car parked itself. The car will eternally remain in this state hereafter, since the car has parked itself and is at rest. In addition to serving as a state machine as described above, moveCar also makes available 2 values – rsDist and rrsDist – to its sub-state machines, detectParking and parkCar. rsDist stores the values of the side distance in the previous clock tick of the moveCar state machine, while rrsDist stores the value 2 clock cycles earlier.

Timing: The moveCar state machine is invoked every 100ms. The moveCar state machine also serves as a clock for the detectParking and parkCar state machines. When in State 1, each clock tick of the moveCar state machine serves as a clock tick for the detectParking machine. When in State 3, each clock tick of the moveCar state machine serves as a clock tick for the parkCar machine.

e) detectParking

Function: The function of detectParking state machine is, as its name suggests, to detect a parking space to park in. It accomplishes this by continuously polling the distance values from the side distance sensor.

Diagram:



Working: detectParking is a 6 state state machine, as can be seen in the diagram above.

State 0 serves as a start-up. This is essential because the first few cycles of the detectParking take place while the side distance sensor is still calibrating itself. Once the wait state is done, the state machine enters state 1.

State 1, essentially, searches for a sudden increase in the side distance value. A sudden increase corresponds to the beginning of a parking space. It does this by checking the (sDistance – rsDist) value. If there is a sudden depression, sDistance will increase and so it's difference from its own previous value (rsDist) will be a large number. When this does occur, the state machine goes onto State 2.

In State 2 it attempts to confirm that it indeed is detecting a valid depression, by calculating (sDistance – rrsDist). Since State 2 is invoked 1 clock tick after the depression was last detected in State 1, rrsDist will store the value of the side distance before the depression began, i.e., from 2 clock cycles earlier. If (side distance – rrsDist) is still a large number, we can confirm that a depression has been detected, and we move to State 3.

In State 3, we keep track of how long the depression is. This is done by incrementing the detect.controlBits for each state machine clock tick that we are still in the depression. When there is a sudden decrease in the value of the side distance, we move to state 4, since it signals a probable end of the parking lot.

State 4 confirms that the possible end of the parking space, as detected in State 3, is indeed the end of the space. This is done in a manner similar to the confirmation done in State 2 using the rrsDist variable.

Once a parking space has been detected by the above states, the state machine moves into State 5 wherein it checks the control Bits (which kept track of how long the parking space was by incrementing for each cock tick while in the depression) to make sure the parking space is large enough. If large enough, then the isParkingLot flag is asserted which would direct moveCar to stop and start the parking sequence.

Timing

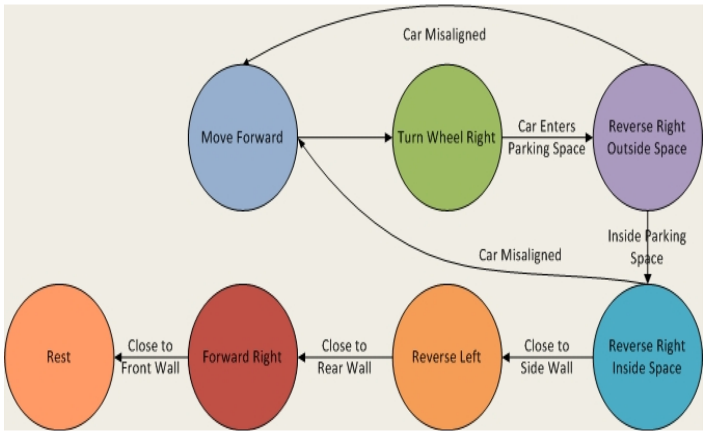
Each tick of the detectParking state machine corresponds to a tick of the moveCar function. When moveCar is in State 1, it calls detectParking on each of its ticks. Therefore, detectParking is called every 100ms until a parking space has been located.

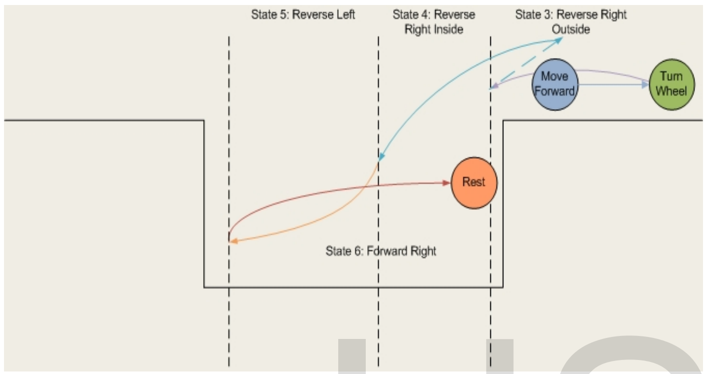
f) parkCar()

Function: The function of the parkCar state machine is to park the car once a parking spot has been identified. The algorithm to park the car continuously interacts with its surroundings through the forward, side and rear sensors.

Working:

The parkCar function tries to simulate how a human would parallel park. It is, essentially, just the following 4 motions:





1. Reverse Right until you are inside the parking lot.
2. Go Forward and redo 1. if the car is not aligned.
3. Reverse Left until the car is fairly straight and close to the back wall.
4. Forward Right until the car is straight and close to the front wall.

The above routine is accomplished using a 7 state machine. State 0 makes the car move forward by a certain amount. The idea is to give the car enough space to move and rotate into the parking space. State 1 simply turns the front wheels to the right. We turn the wheel before reversing the car so as to not lose turning radius by turning as the car reverses.

State 2 commands the car to go reverse right for a specified amount of time until the car has passed the edge of the parking space. Once past the edge of the space, it moves to state 3. In State 3, the car continues in reverse right until it is either a certain distance from inside of the parking space, or the rear distance is close to the edge. These conditions, as can be seen from the figure above, are checks to verify that the car is deep enough inside the parking lot to be able execute the reverse left maneuver. Once the conditions are met, the car stops and the state machine moves to state 4.

- *Ayush Shah is currently pursuing bachelors degree program in electronics and telecommunications engineering in Dwarkadas J. Sanghvi College of Engineering, Mumbai. E-mail: ayushs12@gmail.com*
- *Vidhi Raval is currently pursuing bachelors degree program in electronics and telecommunications engineering in Dwarkadas J. Sanghvi College of Engineering, Mumbai. E-mail: vidhiraval.vr@gmail.com*

NOTE: If at any point in states 1, 2 or 3 the car's AI decides it is not in a position to go through with the parking, it will go back to State 0, and redo the whole procedure.

In State 4, the car moves reverse left. It does this until the rear of the car is close to the side wall of the parking space, which can be judged by the rear distance sensor value. Once close enough to the rear value, it stops and moves to state 5. State 5 commands the car to go forward right. This attempts to straighten out the car completely and to align it nicely inside the spot. It goes forward right until it is close to the side wall of the parking space, as judged by the forward distance sensor. Once aligned, the car is parked and it moves to state 6.

State 6 is a 1 cycle stop before progressing back to state 0. Also, here the isParked variable is set so that the moveCar state machine can move out of parking mode to rest mode.

Timing: Each tick of the parkCar state machine corresponds to a tick of the moveCar function. When moveCar is in State 3, it calls parkCar on each of its ticks. Therefore, parkCar is called very 100ms while the car is being parked.

3. CONCLUSION AND FUTURE SCOPE

Thus we have discussed a very innovative and inexpensive method of autonomous parallel parking. So just with the help of a few IR sensors and a microcontroller that controls the front/back and left/right movement of the wheels by taking inputs from the sensors we have solved the problem of parallel parking. Autonomous parallel parking, if used in the real world will reduce the number of accidents drastically and also make optimum utilization of the space available.

4. REFERENCES

1. http://en.wikipedia.org/wiki/Automatic_parking
2. Jin Xu, Guang Chen and Ming Xie; " Vision- Guided Automatic Parking for Smart Car," Intelligent Vehicles Symposium, 2000. IV 2000, pp. 735 – 730, Oct 2000.
3. Z. L. Wang, C. H. Yang, and T. Y. Guo, "The design of an autonomous parallel parking neuro-fuzzy controller for a car-like mobile robot," in Proceedings of the SICE Annual Conference, Taipei, 2010, pp. 2593-2599.
4. Y. K. Lo, A. B. Rad, C. W. Wong, and M. L. Ho, "Automatic parallel parking," in Intelligent Transportation Systems, 20 03. Proceedings. 2003 IEEE, 2003, pp. 1190-1193 vol.2.