CASE WESTERN RESERVE UNIVERSITY EST. 1826

Jinx

I, certify that the engineering design in the vehicle (original or changes) by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.

5/14/2010

_____

Signature Date

I. Introduction

Jinx, one of the two entries from Case Western Reserve University, is new to the IGVC competition.

II. Overview

Jinx uses a National Instruments cRIO controller based system implemented on the drive train of a motorized wheel chair from Invacare Corp. The cRIO is used for sensory data acquisition from wheel/motor encoders, gyros, and GPS. It will also be used for motor control and for localization computations. Other algorithms and sensors are implemented on an onboard Mac Mini, which includes video and LIDAR. The Mac Mini also serves as a host and interfaces with the cRIO. Figure 1 and Figure 2 provide a functional and visual overview of the hardware.

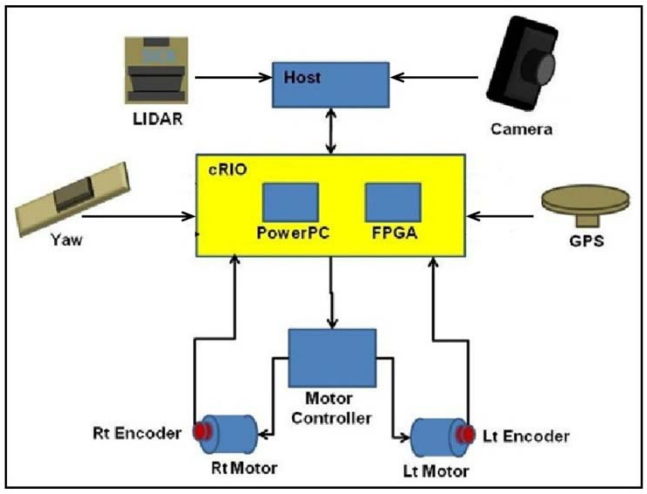

Figure 1: Block Diagram of Jinx

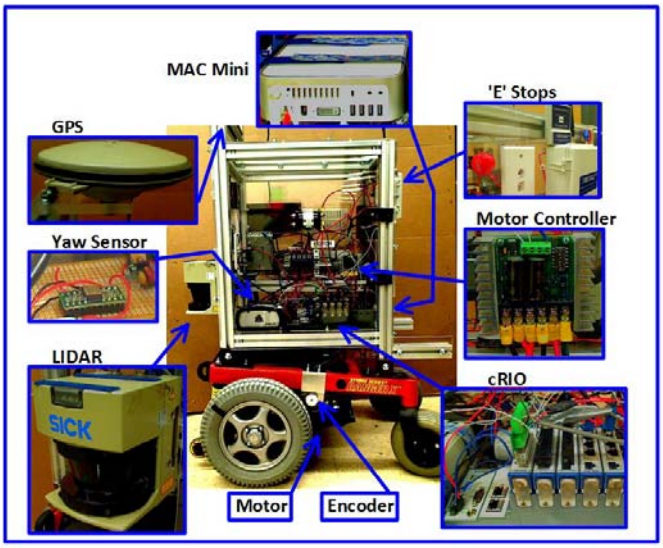

Figure 2: Visual Overview of Jinx

III. Hardware

Drive Train

The base is from the Ranger X model wheel chair from Invacare's Storm series wheel chairs. The wheelchair base provided our team with a rugged base equipped with suspension and motors. The suspension system will help to stabilize the upper frame unit to reduce noise in sensor values, which is a desirable quality for an outdoor robot. Also, since it is a wheelchair base designed for patient care, the base and motors are manufactured so that there is little if any room for failure. The motors provided by the manufacturer are also limited to a 5 mph max speed, which fits our needs perfectly.

Frame

The frame was designed and built using 4 cm square aluminum framing. The frame was built to support two trays filled with electronics. The design was chosen to maximize accessibility and surface area. Designing a method for mounting the frame to our base was one challenge our team had to overcome. As manufactured, the base was sloped down towards the casters. Using a simple mount would cause our frame to also lean at this angle. To overcome this problem our team developed custom mounts that would allow us to level the frame on top of the sloped base.

Electrical System

Jinx's components have many power needs. The LIDARs require a reliable and clean 24V with a low current draw to operate quickly, but the motors require a high current at 24V. The Mac Mini computers require a special 18.5V and other sensors, such as the GPS, require only 12V. To provide this power, the wheelchair base also came with two easily replaceable batteries that are connected in series to give 24V. The base also provided an external port for charging the batteries, making recharging easy.

To generate the other power required for the robot, we used a Carnetix CNX-2140 DC-DC converter to generate 18.5V for the Mac Minis, a Samlex 24-12 DC-DC converter to generate 12V, and a custom-made 24V-24V regulator that produced regulated clean power for the LIDARs. To distribute the power to multiple components, each voltage was fed to a color coded bus bar on separate panels. In our convention, red signifies 24V, blue indicates 12V, and black is used for all grounds. By keeping wiring and bus bars to this coding scheme, our team was able to avoid the costly error of providing sensors with too much power.

For surge protection, our team used a number of different thermal circuit breakers. We used one 120A main breaker that is also used as a main power switch. We also used two 63A breakers, each supplying power to different wheels. For individual electronics, we used small 10A breakers that integrate with the style of bus bar that we chose. A diagram of the power system is shown in Figure 3: Circuit diagram for the electrical system.

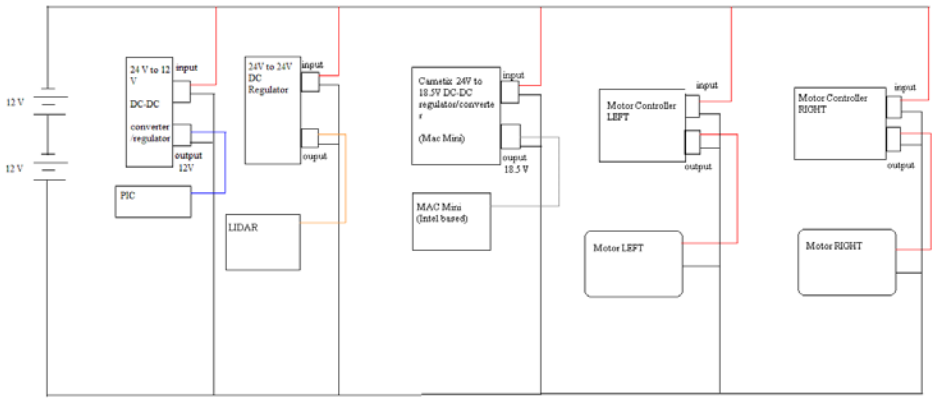

Figure 3: Circuit diagram for the electrical system

The Carnetix and Samlex power supplies were both purchased items that would give us the desired voltage and had reasonable power capabilities. The 24V-24V regulator, however, was custom-assembled by our team. At the heart of the regulator is a store bought regulator that has a range of outputs. Also, since the regulator used is a switching regulator, we needed to filter the output to make sure it was clean power.

IV. Sensors

There are a multitude of sensors on Jinx that are used for physical state observation, obstacle avoidance, and mapping. These sensors include Firefly MV camera, a SICK LIDAR, two wheel encoders, two motor encoders, a GPS receiver, and a yaw sensor.

Encoders

The encoders placed near the wheels are used to determine our robot's velocity for position estimation and we use encoders placed on the motor shafts for PID speed control. Because these encoders are quadrature, we get high resolution and directional information from the two square wave signal outputs. These two signals yield better results than using single direction encoders and needing to guess or have another sensor to determine direction.

Yaw Sensor

The yaw sensor is a device that is capable of measuring Jinx's heading by detecting the relative angular velocity around its vertical axis; this is done by observing changes in the Coriolis acceleration. The sensor used on Jinx is an ADXRS150EB angular rate sensor, which operates at a base rate of 40Hz. The electronic setup on the robot has two separate outputs from the ADXRS150 sensor, a constant 2.5 volt signal and a rate output which delivers a 12.5mV/degree/second signal. The value of the rate output is proportional to the angular velocity of Jinx. By integrating this signal it is possible to calculate the robot's current angle. When Jinx is not in motion the value of the rate output should be approximately 2.5 volts.

GPS

Jinx receives data for its absolute world position by using a Novatel ProPak-V3 receiver. Under the configuration used, the device provides latitude, longitude, heading, and related standard

deviations to the robot. The calculations are made from the United States' GPS constellation with differential error correction provided by OmniStar. The received data consists of three main parts: general header information, a BestPos or BestVel log type, and CRC checksum values sent via binary serial communication from the ProPak-V3 to the FPGA mounted on the cRIO.

LIDAR

Jinx avoids obstacles by the use of SICK Light Detection and Ranging (LIDAR). It uses a rotating mirror to emit an infrared beam within a viewing angle of 180 degrees. With the return of the reflected beam to the LIDAR, the range to obstacles can be detected. SICK has the capability of working in the range of 10 cm to 80 m and can provide data at a rate of 500kbaud. Also, it is an active device as it operates without an external illumination.

Firewire Camera

The Firefly MV from Point Grey Research was used to capture the images. This camera uses a 1/3" wide-VGA CMOS sensor from Micron with a 752 x 480 resolution and a 1394a digital interface. A senko lens was secured to the camera and adjusted to maximize the depth-of-field and provided for the widest angle. This wide angle lens results in image distortion that was corrected using OpenCV library. The images are also captured from the camera using OpenCV.

V. Safety

Throughout the design process safety was a major concern, and was implemented on both the hardware and software side. On the hardware side, three electronic stops were wired into Jinx. These E-stops consisted of a button E-stop on the back of the robot, a wireless E-stop, and lastly a software accessible E-stop. On the software side, there were several soft stops, and speed limiters implemented. The first consisting of a soft stop that was controlled by the LIDAR. If an object was detected inside the LIDARs danger radius, then a stop message is sent to the motor controller to insure that the robot does not hit people or objects. We also added a hardware speed limiter in our FPGA. This ensures that even if a bad motor speed is sent to the FPGA the motor controller will never be sent a dangerous speed.

VI. System Overview

Jinx consists of three separate systems that are each responsible for a subset of the computation required. The FPGA is responsible for real time processing, therefore the PID, and sensory inputs (GPS and Yaw Rate) are processed through here. The cRio runs the physical state observer and steering algorithm. Lastly the Mac Mini reads camera and lidar, and runs the mapper and planner. Figure 4 describes the flow of information for the system, and the location of processes.

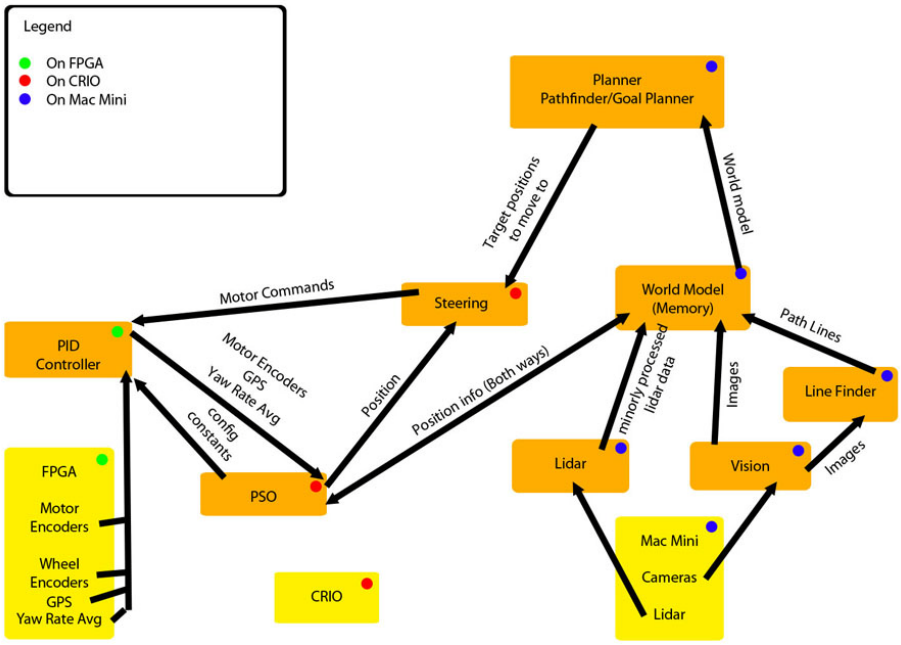

Figure 4: System architecture

VII. Lower Level Software

The lower level software runs on the cRIO, which consists of a FPGA and a PowerPC. The code for the FPGA is written in labview and the PowerPC code is written in C. The cRIO is used for all real-time processing, which includes interfacing with the GPS and yaw sensor, and reading the motor and wheel encoders.

Motion

The motor control consists of two possible paths (Figure 5): the first path, which is used by the robot for most operation, takes speed commands for each motor from the steering module and encoder ticks from the motor encoders to find the error for the PID block. This is used to generate the duty cycle for that goes to the PWM generator for each motor. The alternate path is for the radio control, which is used for manually maneuvering the robot, in the event that higher functions are not operational. In this state, the duty cycle for the PWM generators is determined directly from the PWM signals generated by the Futaba for speed and heading. The entirety of this system resides on the FPGA.

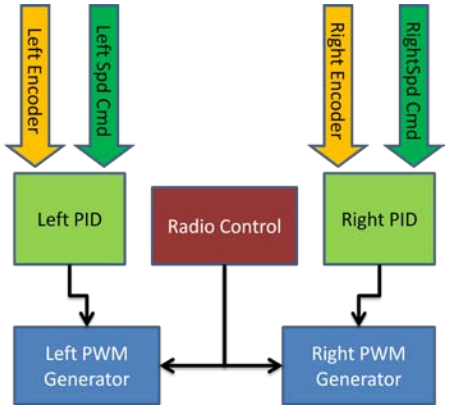

Figure 5: Motor Control Paths

PID Operation

We are using the NI FPGA SubVI as our controller. The block takes as inputs:

- Desired speed from the Steering Module,
- Motor encoder ticks per iteration,
- Maximum and minimum rails for the output,
- PID constants K_i, K_p, K_d,
- Integral error and derivative action resets.

We determine motor ticks per iteration by keeping track of the encoder count each time through the loop and taking the difference from the previous state. The iteration length is defined by Steering Module. This was done in order to be able to set the time period of the loop programmatically. The iteration length is currently set to 10 ms.

The maximum and minimum PID saturation levels are set to -500 and 500. To bring this to the PWM required values of 1000 to 2000, a constant of 1500 is added to the signal, between the control loop and the speed controllers. This approach is necessary, because when the E-Stop is activated, LabVIEW's PID saturates to 0 (i.e. the control loop is broken).

The PID constants are set by the LabVIEW's front panel and are set to:

$$K_i: 0.035 \;\; K_p: 1 \;\; K_d: 0.039$$

These values were determined through use of Zeigler-Nichols for gross estimates and trial and error for fine tuning.

The integral windup and derivative action reset to the PID block is controlled by monitoring the voltage from the E-stops. When any E-stop is engaged, the reset value is asserted.

RC Operation

The FPGA can also receive commands from an RC controller, which allows Jinx to be driven by hand without any other software running. The RC control can be turned off and on using a switch on the back of the robot.

Kalman Filter

The Kalman Filter is a probabilistic algorithm that fuses sensor information and accounts for sensor error in order to produce a "best" guess of estimated parameters. Over time (provided non-zero standard deviations) the filter will converge on the "truth". In the implementation on Jinx, the filter was to estimate the position and the heading. The actual algorithm implemented was not a Kalman filter in the classical sense: it did not use matrices to fold all the sensors together, used nonlinear models, and consisted of a "nested" filter that evaluated whether an update was necessary on the heading parameter.

Steering

The steering currently implemented on Jinx is wagon handle steering. The wagon handle steering method is used to calculate the speed and heading of the robot required to reach its intended destination. The algorithm works by generating a line which intersects both a starting and final position on the robot's anticipated position vector. This technique then computes a

circle, with a radius proportional to the deviation from the intended path, and then determines the location of the two points on the circle which pass through the position vector. The intersection closest to the final position is chosen and the angle between that point and the robot is then used to calculate the desired heading.

The heading and speed commands are then passed on to the steering controls which determine the correct heading course-correct and then dictate the individual wheels in order for the robot move towards its objective. Wagon handle steering has been our predominant steering methodology.

Network Stack

Jinx has a network stack that transports information between the Mac Mini and the PowerPC on the cRIO. The stack uses POSIX sockets to transmit the information across the network, which is connected with a wireless router.

VIII. Upper Level Software

The Mac Mini handles all of the upper level software including reading the camera and LIDAR, and converting them to maps, and all planning.

LIDAR and Reflexive Halt

Our LIDAR reflexive halt is based on a set danger radius. If an object is detected within this radius in our LIDAR scan then the reflexive halt signal is sent to the CRIO. The CRIO will stop and remain stopped until it gets a new waypoint command from the planner. This is the most ideal robust solution because the planner should always analyze the map and determine if the robot should wait for the object to move and then send the same waypoint to the CRIO as it was going to before or the planner could decide to plan around the object that has gotten in the way and change the path that the robot should follow.

Image Processing

The purpose of the image processing is to bring in an image, detect the line, and transform the image into a plan view (discussed in vision mapping). The images are acquired using OpenCV

library, specifically the cvCapture class. The major problem is actually detecting the lines on the grass. Several methods were tried, but our final method was color matching in HSV space. Looking at Figure 6, we see that the lines in the S component of HSV show up darker than any of the grass. Therefore we decided to thresh hold of this low value, and it resulted in a very good line detection method. Then the birds-eye-view transform was used on the image to get an overhead and undistorted image of the lines. This method proved to be a useful and simple solution to the line detection issue. Figure 6 shows the original, S component of HSV, and the threshholded image.

Original *S component of HSV* *Threshholded*

Figure 6

World Map

We have a unique world mapping framework that can take data from a number of devices and combine them with the current PSO state to add that data to a map of the world. We have currently 2 maps, one for LIDAR data and one for vision data. These two maps can be superimposed on each other using our planning algorithms or we can selectively plan off of one or the other.

LIDAR-based Mapping

The LIDAR map is very easily built by drawing walls on an image that correspond to the walls that the pings on the LIDAR hit. This image does not need to be rotated by the robots heading because the positions of the LIDAR pings are pre-rotated and translated by our position and angle. This leads to much less computation as image rotation is computationally intensive. Then the image that the LIDAR is drawn on to is added to a larger map of the known world. At the same time we subtract a constant from the free space between the robot and the walls the

LIDAR hit from the world map. This allows us to forget bad data in areas we thought were blocked but in fact are not.

Vision-based Mapping

The vision map works very similarly to the LIDAR map. We take the birds eye view image of the detected lines from the vision algorithm, and draw it centered in an image. So, it appears as though the robot would be in the center of that image looking forward at the lines we have drawn. Then we rotate that image about its center by whatever our heading angle is. Finally we add that rotated image to our vision world map at a position corresponding to our location. After we do that, we subtract a similarly transformed but non threshholded image of the s space from the HSV image. This subtractor image is scaled by 0.1 to ensure we add new data to the overall map but that we also forget bad old data.

Wall Crawling

Wall crawling is implemented by using 5 masks that are built around the robot. There is a forward mask which is centered on the robot and extends forward from it a short distance. This mask is used to detect if there is an obstacle directly in front of the robot. There are two left pointing masks, one a small distance forward of the other. These masks are to detect if there is a wall on the robots left or if the robot is located at the corner of a wall. The remaining two are left pointing masks of varying distances. These masks are to adjust our distance from the wall. All of these masks are used by doing a simple pixel wise AND operation with the mask and the corresponding world map around the robot. The result of that and is summed. If it the sum is greater than one then we know there was an intersection with that mask and something on the world map. All lit pixels on the world map are obstacles so we know there is an obstacle in that mask. So, for example, for the forward mask, if there is a obstacle intersecting it then the robot knows there is something in front of it and turns to its right in place until there is no longer something in front of it. If there is no intersection with the forward left mask but there is one with the longest centered left mask then we are at a left corner and the robot should turn left. Otherwise if the left forward and long left intersect with something, then there is a wall on the left, and we use the knowledge of whether there is an intersection with the middle and close

left masks to determine if we should move slightly closer or further from the wall to maintain a safe distance. This is the entirety of the algorithm.

Bug

The bug algorithm is built directly on top of the wall crawling algorithm. Initially it is given a goal point. It then heads towards that goal point. If there is an obstacle in front of it then it will stop heading towards the goal. It will draw a line on an internal image from a small ways ahead of its position towards the goal. It has an internal mask that represents the robot itself that is a small circle. It will then use the wall crawling algorithm to circle around the obstacle until the robot mask intersects the goal line it drew. Once it intersects with this line then the robot heads directly towards the goal again until it hits another obstacle.

Path Following

The overall plan for the path following is to combine the lidar map objects, and the vision map of lines to make a world map. This map will indicate where objects, clear space, and unexplored territory are in our world. Then to navigate this world map a modified bug algorithm will be used, which will continually be updating its goal point to navigate the circular course. The major obstacle to this method is that the vision map contains discrete lines, which will allow the bug algorithm to navigate itself out of the course. To correct this issue the vision map will be modified to only contain continuous lines by running a virtual axle over the vision map. The virtual axle will have a taunt spring attaching the two wheels together with a width of about ten feet, the distance between the lines. The tires are translated and rotated in their current position until a threshold value of line pixels are beneath them. The current orientation and position of the tires will be marked on a separate map, and then the tires will be moved forward. The same process is repeated, until either one or both tires cannot find enough pixels to call the current orientation the line. If one tire cannot find enough pixels, then the axle comes into action by allowing the known tire to guide the tire that cannot find a line. Therefore, a line will be generated on both sides of the lane. The case where neither tire can find a line is a much harder situation to deal with. Each tire will mark their current location, the last known line location, and continue to search for a line until a line is found. The tires will

keep the same general trajectory, but as they get farther and farther away from the last known location the search area will grow. Once a line is found the path will be backtracked, and the line filled in. This technique will again give a continuous line to follow.

IX. Expected Performance

We expect the robot to travel at speeds between 0 and 1.5 meters per second. It also should have no difficulty going up or down ramps. The highest-level software loop runs at 15 hertz so the robots reaction time should be no more than a quarter of a second, generously. The lidar scanning range is set to 80 meters, so objects are detected within that 80 meters. Our navigational waypoints are accurate to within 20 cm, provided we have clean GPS data. The battery life on the robot is 2 hours. Our planning algorithm should successfully deal with all complex obstacles.

X. Cost

Part Description	**Retail Price**	**Cost to team**
Wheelchair base	$3,000	$0
Batteries	$400	$400
Bosch Aluminum Framing	$500	$500
Power Converters	$345	$345
Electrical Components	$770	$770
Wireless EStop	$490	$490
Sabertooth Speed Controller 2x25	$125	$125
SICK LIDAR	$6,000	$6,000
Network Equipment	$40	$40
Yaw Rate Gyro	$80	$80
Quadrature Encoders	$320	$320
RS422-USB Converter	$80	$80
FireFly Camera	$350	$350
Camera Lens	$150	$150
RC Controls	$500	$500
NovaTel ProPak GPS	$5,490	$2,700
cRIO and Modules	$4,300	$4,300

Total	$22,940.00	$17,150.00