Abstract

of

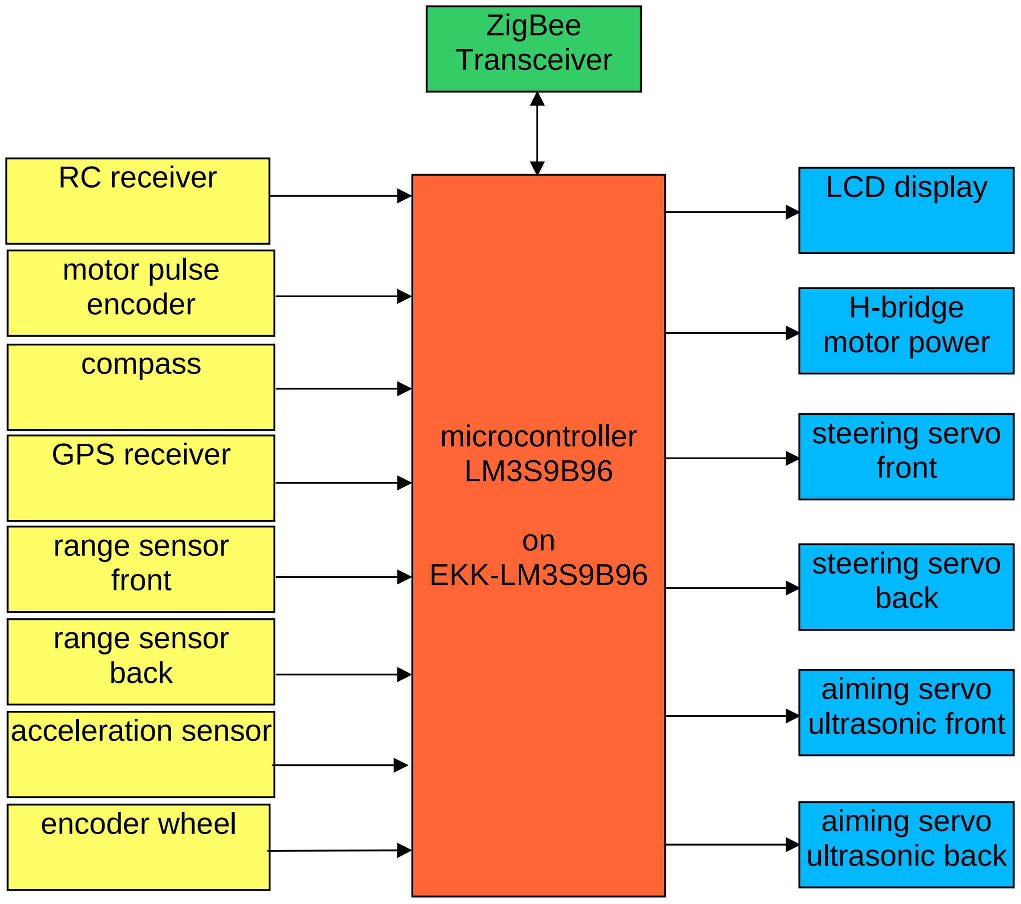
Entry TI2827

“Crawler”

for

Design Stellaris 2010 competition

Subject of this project is an autonomous robot, equipped with various sensors, which moves around the environment, exploring it and avoiding obstacles. The goal was to develop a robot platform that leaves room for further experiments and improvements. To be of use in the real world, like e.g. as lawn mower, cleaning machine or robo-magellan contestant, the robot is based on a quite big, commercially available 1:8 scale RC model rock crawler. Both axes are build exactly symmetric, each with a steering servo, a motor and a self-locking differential. The axes are suspended very flexibly to allow nearly 90 degree articulation which makes the vehicle able for hard off-road terrain.
The RC control's receiver is still included to provide a manual control mode for test purposes. More interesting are, of course, the automatic modes where the robot drives autonomously under control of the LM3S9B96 microcontroller, which – with its multitude of peripherals - showed to be highly suitable for such a robot controller. The following block diagram shows how the sensor signals are used by the microcontroller to drive the crawler's motors and servos:

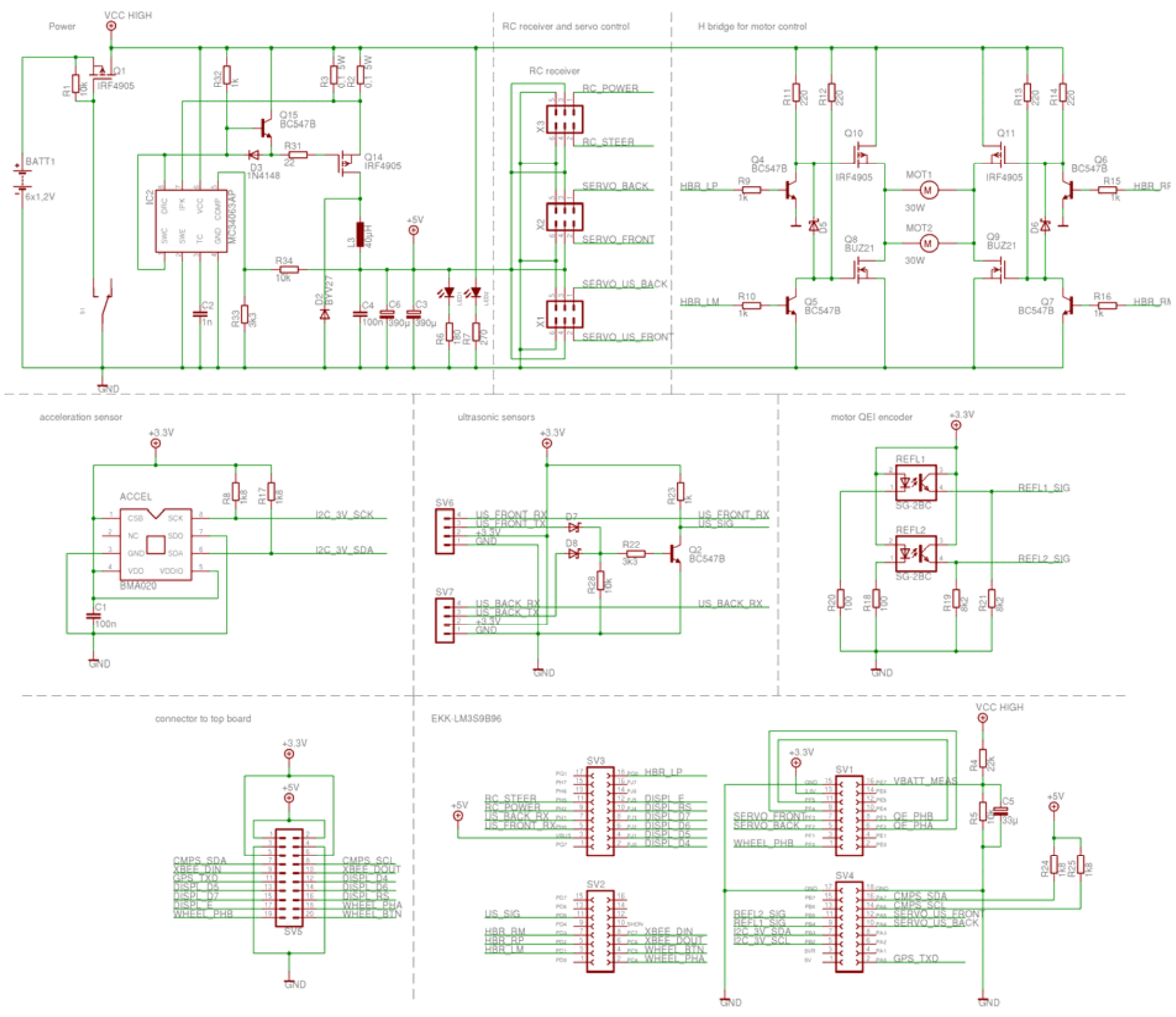


By means of counting the motor pulses and reading the compass module, the microcontroller computes its current position via odometry. By interpretation of the GPS position or known obstacles, it could try to improve the precision. Unfortunately the GPS coordinates are not used for navigation, yet.
Both range sensors are ultrasonic distance sensors that are mounted on micro servos and can therefore be tilt nearly 180 degrees to find out where to best try to drive around obstacles. It would also be possible to add a control mode where the robot would follow a wall by swinging the back sensor to nearly 90 degrees and control the steering to keep the distance to the wall constant.

The acceleration sensor can be used to detect steep slopes where the robot is in danger to overturn. The robot could react by driving back or at least performing an emergency stop.
By means of a ZigBee module the robot sends continually its state to a PC where it is displayed to the robot owner with the JAVA application “CrawlerControl”. This application also shows the 2-dimensional track of where the robot has driven. It also allows to change the control mode of the robot and some other parameters by sending commands to the robot.
In the beginning it was planned to have just this PC application to control the robot, but it turned out that it would also be quite handy if the robot had some kind of user interface, too, to be able to at least display some important values like battery state, GPS information and compass readout. Therefore the robot got a simple user interface consisting of an encoder wheel to be able to navigate through menus and a 2 lines by 16 columns LCD display. The menues are not implemented yet, but the wheel allows to scroll through all interesting state and measurement values.
The motors for both axes are driven by a H-bridge with PWM for fine power control. It also allows to brake by short circuiting the motors.
Two servo outputs operate the steering servos of the axes. Two other servo outputs operate the micro servos for tilting the ultrasonic sensors.

By taking advantage of the peripherals of the microcontroller, it was possible to get along with comparatively few external electronic components, except of course of the power supply and the H-bridge for the motors.
The electronics are distributed over two boards. The lower one is the main board with power supply, H-bridge, servo outputs and microcontroller module. The top board hosts the user interface and the interference-prone compass and GPS receiver.

The following schedule shows the main board:



The software is written as a collection of SafeRTOS tasks which represent the data flow from the sensors over the control to the outputs. The “DriveSensTask”, for example, reads out the compass and the motor pulse encoder and sends the measurements to the “PositionTask”, which calculates the robot's position and sends it to the “ControlTask”. This task decides on the steering and motor power and sends a corresponding message to the “DriveTask” which finally sets the outputs. If necessary, the DriveTask could change the motor power gradually to avoid wheel slippage.

Below is a small code example of the PositionTask which shows a typical task function, where the task waits for a message, reacts to it and sends its result to the next task:

```
/**
 * Task function that reacts to incoming messages and sends out position messages.
 */
static void PositionTask(void *pvParameters)
{
    // Loop forever.
    while(1)
    {
        /** structure to receive and send messages from/to other tasks */
        Message_t    msg;

        /** flag, if the msg is really to be sent out */
        int            flagSend = 0;

        // wait for message
        if (xQueueReceive(gPositionTaskQueue, &msg, portMAX_DELAY) == pdPASS)
        {
            // successful, so check which message was received and compute new position
            switch (msg.type)
            {
                case MSG_TYPE_DRIVESENS:
                    // compute position from drive sensors.
                    // TODO: sensor fusion with GPS
                    updateDriveSens(&msg);

                       // because the position might have changed, we send a position message
                    flagSend = 1;
                    break;
            }

            if (flagSend)
            {
                msg.type = MSG_TYPE_POSITION;
                msg.u.position.x = (int) (posX * 1000);        // transform to millimeters
                msg.u.position.y = (int) (posY * 1000);        // transform to millimeters
                msg.u.position.dir = dirDeg;

                // send message to control task
                xQueueSend(gControlTaskQueue, &msg, MSG_WAIT_SEND_STD);

                // send message to comm task
                xQueueSend(gCommTaskQueue, &msg, MSG_WAIT_SEND_LOG);
            }
        }
    }
}
```