Vex Robotics Tutorial
EECS 690: Robot Intelligence

Christopher M. Gifford
Ph.D. Student, Computer Science
The Robotics Group, CReSIS
cgifford@cresis.ku.edu

September 11, 2007
Overview

- Past Robots for Course
- Vex Robotics Design System
  - Contents of Robot Kit
  - Overview of Basic Features
  - Available Sensors
  - Example Robot: Squarebot
- easyC Development
  - Interface
  - API and Programming Overview
  - Downloading Code to Robot
- Troubleshooting
- Resources
Past Robots for Course

- **Rug Warrior Pro Mobile Robot**
  - Interactive C (IC)
  - Shaft encoders, 2 wheels, skirt
  - LCD debugging
  - 32K memory
Past Robots for Course
Past Robots for Course

- Palm Pilot Robot Kit (PPRK)
  - BrainStem C
  - Tiny Embedded Application (TEA)
  - 3 omnidirectional wheels
  - Limited EEPROM and slots

http://www.acroname.com/technology/103/abstract.html
Vex Robotics Design System

- Inventor’s Guide
- Starter Kit
- Sensors and Subsystems
- Microcontroller Specifications
- Programming Kit
- easyC Development
  - User Interface
  - User API
  - Compiling and Downloading Code
- Sample and Testing Programs
- Debugging
- Troubleshooting
- Resources
- Example Vex Robots
Supports several skill levels
Covers everything in your starter kit
Broken up into several subsystems
- Structure
- Motion
- Power
- Sensors
- Logic
- Control
- Programming
Instructions to build, program, and operate Squarebot
Troubleshooting
Resources
Extra: 2 linear sliders, 2 ultrasonic range finders, 2 light sensors
Structure: Stability

- Structure is very important in robot design
- Should design robot for the expected environment and task
- Sensing should be taken into account during design as well
  - What sensors can help accomplish the task?
  - How and where do those sensors fit into/onto the robot?

- Center of gravity
  - Average of both weight and position on the robot
  - Heavier objects count more than lighter ones
  - Pieces further out count more as well

- Support polygon
  - Formed by connecting points where robot touches the ground
  - There is always one support polygon in any configuration
**Stability**

- Most stable when center of gravity is centered over support polygon
- Robot will topple over if center of gravity falls outside support polygon
- Gripping and moving objects alters center of gravity WRT support polygon
- Adding weights or larger support polygon larger can help offset changes
Structure: Sturdiness and Vulnerability

- **Sturdiness and stress**
  - There are over 100 screws in the kit, so use them
  - Secure parts together well using multiple screws, if necessary
  - If you don’t want something to rotate, use two screws
  - More weight (especially suspended) strains the mounting point
  - Bracing heavy or long parts can help provide support to reduce strain

- **Vulnerability**
  - You will be running into things
  - Protect cables, microcontroller, crystal, and volatile components from
    - Collisions
    - Getting caught on something
    - Being run over
  - Protecting the sensors and technology from the environment, obstacles, or other bloodthirsty robots can help increase your lifetime and reliability
Motion: Motors

- **Motors and servomotors**
  - Motors transform electrical into mechanical energy
  - Electrical power converted to physical motion
  - Spin in opposite directions due to internal motor designs
  - Clutch: protect internal gearing from damage (breaks connection)

- **Standard motors (3 in kit)**
  - Spin axle around completely and keep going
  - Should be used whenever continuous motion is needed
  - Example: drive system

- **Servomotors (1 in kit)**
  - Turn axle to face specific direction within range of motion (120° in Vex kit)
  - Should be used where boundaries of motion are well-defined and where specific positions need upheld within these boundaries
  - Example: open/close gripper
Motor generates power (specific amount of energy per second)

Trade-off between torque and speed with set amount of energy to go around

- **Torque**: force with which motor can turn the wheel
- **Speed**: rate at which motor can turn the wheel
- Speed-torque balance shifts based on different combinations of gears between motor and wheels

**Gear ratio**

- A multiplier on torque and divider on speed
- **Driving gear**: provides force to turn other gear, usually attached to motor
- **Driven gear**: gear being driven by the driving gear
- **Gear ratio** = Driven_Gear_Teeth / Driving_Gear_Teeth
- Example: 1:3 has 1/3 as much torque as 1:1 but 3 times as much speed
Motion: Gears

- **Idler gear**
  - Gear between driving and driven gear
  - Has no effect on gear ratio (cancel out)
  - Reverse direction of spin for each idler gear
  - Can also be used to transmit force over a distance to another gear

- **Compound gear ratio**
  - One or more pairs of gears share an axle
  - Compound gears allow force and speed configurations not normally achieved with available components
  - Calculated by multiplying the gear ratios of each individual gear pair
  - Example: $12:60 \times 12:60 = 1:5 \times 1:5 = 1:25$
    - Turning axle 25x faster with $\frac{1}{25^{th}}$ the force
Motion: Wheels

- **Wheel size effects robot's acceleration and top speed**
  - Bigger tires provide slower acceleration but a faster top speed
  - Larger wheels cover more ground with same rotation
  - Smaller tires provide faster acceleration but a slower top speed
  - Higher gear ratios may take slightly longer to reach top speed

- **Wheels convert torque into a pushing force on the ground**
  - Force = torque/(distance from center to edge of wheel)
  - Smaller wheels = larger pushing force = faster acceleration
  - Larger wheels produce a smaller amount of force for same torque

- **Traction and friction**
  - Friction dissipates some of the robot's energy
  - More friction: wider, bumpier, or stickier tires
  - Less friction: narrower, smoother, or more slippery tires

- **Design decision based on task, terrain, and robot structure**
Vex robots use rechargeable batteries for an energy source

- Robot: 6 AA (7.2 V), Transmitter: 8 AA (9.6 V), stacked in series
- Power pack: charges both packs, auto power-off and protection
- Microcontroller: green (OK), red (need recharging)
- Transmitter: 9.4 V (low), 8.9 V (very low, < 10 min), 8.5 V (stop!)

NiCd (Nickel-Cadmium chemical composition)

- Rechargeable; more energy than comparable AA batteries
- Provide constant reliable voltage until exhausted
- No permanent memory effect for modern batteries (Vex tested)
- Charging: Transmitter: 1.4-2.0 hrs, Robot: 1.4-2.8 hrs
Memory effect
- Battery capacity permanently diminishes by not fully discharging each time
- Said to be less of an issue in modern consumer devices

Voltage drop
- Shallow-discharged repeatedly results in lower voltage
- Curable: run completely down (device turns off) and charge fully again

Notes on batteries
- Charge fully the first time
- Fresh batteries are important for competitions
- Calibrate robot with identical batteries that will be used in competitions
- Don’t use Alkalines
  - Cannot provide power fast enough
  - Provide decreasing voltage as they are used up
  - Rechargeables lose power with each recharge
Provide robots a way to measure things about its environment

- Depending on the sensor suite and context, this can tell many things

**Analog vs. Digital**

- **Analog**
  - Voltage measure between 0 and maximum range
  - Difficult to send/maintain specific voltage in noisy environments
  - Example: light sensor: 0 V (dark), max V (very bright), between (some light)
  - Vex function calls: return between 0 and 100 or 1024

- **Digital**
  - Rounded voltage to low (0 V) or high (max V), no in-between
  - Reliable in noisy environments due to rounding
  - Example: bump sensor: 0 V (pressed), max V (not pressed)
  - Vex function calls: return 0 or 1
Sensors: Bump Switch Sensor

- Physical switch digital sensor (not pressed or pressed)
- Not pressed: high signal (1)
- Pressed: low signal (0)
- Can be placed anywhere in Analog/Digital ports 1-12
- Port must be configured as Digital Input
- Code
  - `bump = GetDigitalInput(port); // Returns ‘unsigned char’`
- Total: 2
Sensors: Optical Shaft Encoder

- Digital sensor involving IR light sensor and IR LED
- Sense disk: high signal (1), Not sense disk: low signal (0)
- As disk rotates, encoder generates a string of signals
  - Example: 0101010, can be counted to determine amount of rotation
  - One full revolution (360°) of the encoder is equal to 90 encoder pulses
- Can be connected to any Interrupt Port (1-6)

**Code**

- PresetEncoder(interrupt_port, count_preset); // Preset encoder value
- StartEncoder(interrupt_port); // Start encoder counting
- encoder = GetEncoder(interrupt_port); // Returns ‘unsigned long’
- StopEncoder(interrupt_port); // Stop encoder counting
Physical switch digital sensor (not pressed or pressed)
- Not pressed: high signal (1)
- Pressed: low signal (0)
- Can be placed anywhere in Analog/Digital ports 1-12
- Port must be configured as Digital Input

Code
- `limit = GetDigitalInput(port); // Returns ‘unsigned char’`

Total: 2
Sensors: Light Sensor

- Analog sensor (feedback range between 0 and 1024)
- Photoresistor: darker translates to higher numerical value
- Very bright = 0, Some light = 512, Very dark = 1024
- Range of 0 to 6 ft (dependent on ambient light and light source)
- Can be placed anywhere in Analog/Digital ports 1-12
- Port must be configured as Analog Input
- Code
  - `light = GetAnalogInput(port); // Returns ‘unsigned int’`
- Total: 2
Sensors: Line Tracker Sensor

- Analog sensor (feedback range between 0 and 1024)
- IR light sensor and LED: darker translates to higher value
- Very bright = 0, Some light = 512, Very dark = 1024
- Line width: 0.25 inches minimum, optimal line width of 0.5 inches
- Optimal range: 3 mm, effectiveness drops off by factor of 10 at 5/8"
- Can be placed anywhere in Analog/Digital ports 1-12
- Port must be configured as Analog Input
- Code
  
  ```
  line = GetAnalogInput(port); // Returns ‘unsigned int’
  ```
Sensors: Ultrasonic Range Sensor

- Produces analog signals (ranging from object close to not close)
- Uses high frequency sound waves to detect objects
  - Time translated into numerical value between 2 (closer) and 100 (furthest)
  - Can determine distance to an object between 3 cm and 3 m away
  - Pulses at 40 KHz for 10 μsec and receives at 40 KHz
- Input wire to Digital Output port, output wire to Interrupt port
- Code
  - StartUltrasonic(interrupt_port, digital_out_port); // Start recording waves
  - GetUltrasonic(interrupt_port, digital_out_port); // Returns ‘unsigned int’
  - StopUltrasonic(interrupt_port, digital_out_port); // Stop recording waves
- Total: 2
Robots have 2 advantages over other mechanical systems

- Can sense important things about the environment
- Can process sensor information and react/behave/reason intelligently

Vex microcontroller coordinates flow of info and power on robot

- 2 transmitter ports (Rx1, Rx2), Serial port (programming), and power
- 8 motor ports and 6 interrupt ports
- Digital/Analog ports: 1-12 (sensors), 13-16 (jumpers), TX and RX
- All connections keyed
Logic: Microcontroller Specifications

❖ Performance
  • Digital input frequency: 50 KHz
  • Analog input access: 10 μsec
  • User microcontroller: Microchip PICmicro® PIC18F8520
  • Processor speed: 10 MIPS (Million Instructions Per Second)

❖ Programming
  • Language: PIC C
  • Program space: 32 K words = approximately 128 KB (hex file)
  • RAM: approximately 2 KB, for memory-mapped I/O and PIC devices
  • EEPROM: approximately 1 KB, for data memory
  • Programming tools: Microchip MPLAB IDE, easyC, or text-editor/Makefile
  • C18 Compiler

❖ Comes with default programmed behavior (discussed later)
Logic: Example

- **Cliffbot**
  - Uses limit sensor to detect cliff and react
  - Disables user control and can turn itself around and return control
  - Simple RC car would just plummet to its death!

Simple RC Car
Control

- Link between human operator and robot with RF transmitter
- Command sent via FM radio waves to RF receiver on robot
  - Operates on 75 MHz band (75.410 MHz, aka Ch. 61)
  - FM: combines basis wave (carrier) and modulating wave (signal → data)
  - Transmitter and robot channels must match to communicate
  - Need different channels for each robot for manual robot competitions
  - Warning: turn transmitter on before robot (interpret stray radio waves)
- Radio waves radiate out of side of antenna
  - Best range and performance if not pointing directly at robot
- 2 ports on microcontroller for use of 2 transmitters at same time
  - Rx1 (default) and Rx2
  - Example: one controls driving, another controls gripper
  - Starter kit only has 1 transmitter, however
**Control**

- **Tether port on back of transmitter**
  - Connect directly to microcontroller with phone cable
  - Diagnostic purposes (radio interference or code the problem?)

- **Driving modes**
  - *Tank style*: “23” mode (default)
    - Right stick: motor port 2
    - Left stick: motor port 3
  - *Arcade style*: “12” mode
    - Uses only right joystick for throttling and turning
    - Horizontal axis: control channel 1
    - Vertical axis: control channel 2

- **Inventor’s Guide**
  - Operation and settings for transmitter
  - Calibrating, trimming, and scaling the sticks
Programming

- **Analyze problem at hand and define desired robot behaviors**
  - What sensors and behaviors are needed?
  - Break behaviors down into programmable parts
  - Many ways to solve a problem
  - Attempt to design and program the best solution for your situation

- **Vex Programming Kit**
  - easyC software
  - USB-to-Serial cable
  - Programming module
  - Used to develop and download programs from computer to robot’s microcontroller

- **Inventor’s Guide covers easyC software installation**
Programming: easyC Overview

- **Graphical programming environment**
  - Visual block diagram and corresponding code is filled in
  - Can only edit visual blocks and their fields, not the actual code
  - Drag function block icons into place and fill in appropriate fields
  - Levels L1, L2, PRO: determine what functions are available to use
  - Some Visual C++ or Visual Basic feel to the IDE

- **C syntax**
  - Program flow: while, for, do while, if, else, comparisons, assignments
  - Pointers, passing by reference, pass by value, etc
  - Can use // or /* … */ for comments
  - MUST declare variables at top of function or globally

- **Programs consist of .BCP and .ECP (easyC Project Session File)**
  - Must move both if relocating program
Programming: easyC Interface

```c
void main ( void )
{
    //Connect Encoder to Interrupt 2
    //There are 90 encoder counts per revolution
    PresetEncoder (2, 0);
    StartEncoder (2);
    while (loop == 1)
    {
        encoder = GetEncoder (2);
        PrintToScreen ("encoder =%d\n", (int)encoder);
    }
    //Connect Encoder to Interrupt 2
    //There are 90 encoder counts per revolution
    PresetEncoder (2, 0);
    StartEncoder (2);
    while (loop == 1)
    {
        encoder = GetEncoder (2);
        PrintToScreen ("encoder =%d\n", (int)encoder);
    }
}
```

For Loop

Expression:

```c
for ( ; ; )
```

Add Variable: ▼ Add Operator: ▼

Code:
```c
for ( ; ; )
```

Comment:
Configuring the port bank
- Double-click “Config” block on easyC screen
- In Analog/Digital section
  - Left-clicking changes between Digital Input and Output
  - Right-clicking changes between Analog and Digital
    - Electrical load limitations make Analog Output not possible
  - Analog ports must all be in a block

All ports are 5 V in or out (no more!!!)

- Digital Output: ultrasonic range sensor, LEDs, other 5 V outputs
  - SetDigitalOutput(port, value);
- Digital Input: bump, limit, jumpers, other 5 V sensors
  - GetDigitalInput(port);
- Analog Input: light, line tracker, other 5 V analog sensors
  - GetAnalogInput(port);
Programming: easyC Functions

- Create your own functions

- Create your own code
Programming: easyC Functions

- `PrintToScreen("Bumper Switch = %d\n", (int)bump);`

- **Timing**
  - `void Wait(msecs)`
  - `void StartTimer(unsigned char ucTimerNum);`
  - `void PresetTimer(unsigned char ucTimerNum, unsigned long lValue);`
  - `unsigned long GetTimer(unsigned char ucTimerNum);`
  - `void StopTimer(unsigned char ucTimerNum);`

- **More UserAPI.h functions**
  - `Arcade2, Arcade4`
  - `Tank2, Tank4`
  - `MotorRcControl`
  - `ServoRcControl`
  - `GetRxInput`
For two motor (Squarebot-like) setup
- Motor directions must be opposite to drive F or R
- Motor directions must be same for turning L or R

Motor values (0-255)
- 0 (spin fastest CCW), 127 (stop), 255 (spin fastest CW)

Servomotor values (0-255)
- 0 (spin furthest CW), 127 (60°), 255 (spin furthest CCW)

Code
- void SetMotor(port, speed&dir);
- void SetServo(port, position);
- void SetPWM(port, pwm_value);

Turning
- Set motors, wait for an amount of time, then reset motors
- Inventor’s Guide: 500 msec is about 90°
- Advice: test with robot in actual environment and surface (variations)
Programming: Downloading Code

- **Connect robot to computer**
  - Connect USB-to-Serial cable, programming module, and phone cable
  - Connect from USB of computer to “Serial” connection on microcontroller

- **Build & Download in easyC IDE**
  - Compiles code, generating HEX file
  - Downloads HEX file to microcontroller
  - Program runs immediately after downloading
**Programming: Downloading Code**

- **IFI/intelitek Loader**
  - Downloads HEX file to microcontroller
  - C:\Program Files\Intelitek\easyC for Vex\Loader\iLoader.exe
  - easyC: Build & Download > Loader Setup…
    - COM port (look in Device Manager for port involving ProlificUSB)
    - Choose what to launch after code is downloaded (terminal)
  - Loader itself: Port Setting and Options menus
  - Program runs immediately after downloading

- **Old code is replaced with new code upon downloading**
Programming: Fresh Starts

- **Download latest (highest version) Master Code**
  - Loader: Options > Download Master Code…
  - Microcontroller’s master code

- **Download Default Code**
  - easyC: Build & Download > Download Default Code
  - Mixed mode operation: autonomous and remote control
  - Safety measure: lose control for 3 sec when bumper pressed
  - Default behaviors discussed in more detail in Inventor’s Guide

- **Download On-line Code**
  - easyC: Build & Download > On-Line Window…
  - Download button downloads on-line code to microcontroller
  - Operate and test each sensor in same window
  - Great debugging and verification tool
Programming: On-Line Code

- Download the code to the Vex microcontroller
- Sensor values automatically updated
- Test individual sensors
- Test sensor limits
- Debug sensor issues
- Control several motors and servomotors
Sample programs
- Location: C:\Program Files\Intelitek\easyC for Vex\Projects\Samples
- Mixed control, 2 RX on same robot, arm limit, flow control
- Transmitter test, ball gatherer

Testing programs
- Location: C:\Program Files\Intelitek\easyC for Vex\Projects\Test Code
- Motor test program
- Individual test programs for each sensor

Template programs
- Location: C:\Program Files\Intelitek\easyC for Vex\Templates
- Program layouts
- General and competition
Visual programming can be slow and strange

Manually-editable C code for Vex
- Available on [my class resource page](#)
- `Makefile` and `vex_c_example.c` files

Compiling code
- Using make-compatible terminal, execute ‘make’ in source directory
- Example: MinGW or cygwin
- This produces the HEX file for loading onto microcontroller
- Compile errors and warnings produced (see Troubleshooting slide)

Download code using IFI Loader
- `C:\Program Files\Intelitek\easyC for Vex\Loader\iLoader.exe`
- Download HEX file produced from compilation
- **Makefile must be slightly modified for project and installation**
  - PROJ is the name of the primary `.c` file you want to compile
    - PROJ = `vex_c_example`
  - EC is the easyC installation directory
    - EC = `C:/Program Files/Intelitek/easyC\ for\ Vex/`

- **C module must contain two functions**
  - void `main(void)` // Program entry point
  - void `IO_Initiation(void)`
    - DefineControllerIO(…) // Port configuration function

- **Notes**
  - All variable declarations must be at top of function or global
  - Complete path to C file must be 62 characters or less (for converter)
  - Example Vex C file drives straightforward until bumper is pressed
  - Demonstrates simple motor and sensor operation
  - Demonstrates repeatedly printing a simple internal map when done
Programming: Manual Edit and Compilation

```
$ make
C:/Program Files/IntelleC/easyC\ for\ Vex//mcc18/bin/mcc18.exe -p18F8520 -iC:/Program Files/IntelleC/easyC\ for\ Vex//mcc18\h -iC:/Program Files/IntelleC/easyC\ for\ Vex//Vex/UserAPI\ -Oi+ vex_c_example.c
MPLAB C18 v2.40 (feature limited)
Copyright 1999-2004 Microchip Technology Inc.
This version of MPLAB C18 does not support the extended mode and will not perform all optimizations. To purchase a full copy of MPLAB C18, please contact your local distributor or visit buy.microchip.com.
WARNING: This version of MPLAB C18 does not support procedural abstraction. Procedural abstraction will not be run.
C:/Program Files/IntelleC/easyC\ for\ Vex//mcc18\bin\mplink.exe -iC:/Program Files/IntelleC/easyC\ for\ Vex//mcc18\lib -iC:/Program Files/IntelleC/easyC\ for\ Vex//Vex/Library -iC:/Program Files/IntelleC/easyC\ for\ Vex//Vex/Object -ouex_c_example C:/Program Files/IntelleC/easyC\ for\ Vex//Vex/\Linker/18f8520 user.lkr vex_c_example.o ifi_startup.o ifi_utilities.o printf_lib.o Start.o user_routines.o interrupts.o user_routines_fast.o user_api.o Vex_alltimers.lib
MPLINK 3.90, Linker
Copyright (c) 2004 Microchip Technology Inc.
Errors
MP2COD 3.90, COFF to COD File Converter
Copyright (c) 2004 Microchip Technology Inc.
Errors
MP2HEX 3.90, COFF to HEX File Converter
Copyright (c) 2004 Microchip Technology Inc.
Errors
```

August 30, 2007
Christopher M. Gifford
SolidWorks Modeling

- SolidWorks Student Design Kit
- DWGeditor and eDrawings
Debugging

- **Use terminal in loader for looking at output**
  - Must be connected to robot using programming cable
  - Can use print statements to output values or behavior status changes
  - This is how we will get information from your robot for competitions

- **On-line code**
  - Allows control of motors
  - Monitors sensor values on robot directly from computer
  - Valuable for testing and troubleshooting

- **Capturing entire sessions of output and store to a file**
  - Use RealTerm or similar to monitor COM port
  - Direct data to file for storage and later analysis

- `PrintToScreen("Bumper Switch = %d\n", (int)bump);`
- **Tether port on back of transmitter**
Troubleshooting

- **Compiler provides error, line number, and brief description**
  - “Symbol has not been defined”
    - Misspelled variable name or used a variable not yet defined
  - “Syntax error”
    - General C syntax or didn’t finish loop condition(s)
  - “Expression is always true”
    - ELSE never entered due to IF condition always being true
  - “___ name exceeds file format maximum of 62 characters”
    - Move source directory to a location with a shorter full path
  - “To enable download …”
    - Commonly: COM port being used by another application or using wrong port
  - “Cannot write C File” and “Access to ____ was denied”
    - Privileges and access rights for user on system

- **Sensor not working?**
  - Make sure you are using the correct ports (Analog/Digital/Motor/Interrupt)
  - Make sure the ports are configured for your mode (Input or Output)
easyC
- Help > Help
- Help button for each function block and sensor

Inventor’s Guide

Internet
- VexLabs
- Vex Robotics Downloads
- Vex Forum
- Vex Example Code
- Vex Robot Photos
- Vex Tutorial with SolidWorks
- Vex Microcontroller Specifications

Me
- Class resource page
- cgifford@cresis.ku.edu
Example Vex Robot Designs
Thank You

Questions?

Office Hours: 1:00-2:00PM TR
Email: cgifford@cresis.ku.edu