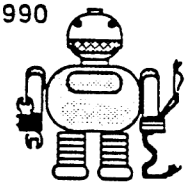
August 1990

ROBOT BUILDER

The official publication of the Robotics Society of Southern California
P.O. Box 3227, Seal Beach CA 90740, Meetings the 1st Tuesday @ 7:00 PM at MTI College



UPCOMING EVENT CALENDAR:

July 29:	Computer Swap Meet at Advanced Computer Products
August 7:	RSSC August Meeting, MTI College: Topic - TBD
August 11:	RSSC Robot Project Workshop, *The Robot Company*
September 4:	RSSC September Meeting, MTI College: Topic - TBD
September 8:	RSSC Robot Project Workshop, *The Robot Company*
October 2:	RSSC October Meeting, MTI College: Topic - TBD
October 6:	RSSC Robot Project Workshop, *The Robot Company*

JULY 3rd RSSC MEETING

We had another good turnout at our July 3rd meeting with about 22 people in attendance. Our featured speaker for the evening, Craig Rubenstein from *GMF Robotics*, was very informative. Craig discussed his company's activities in the industrial robotics industry and the types of robotics systems they design and manufacture. In addition, he presented a videotape showing some of their industrial robots in action and demonstrating their capabilities.

The beginning of the meeting was led by Tom Carroll, who discussed a variety of RSSC business items. These included (1) staffing the OC Fair booth, (2) staffing the ACP Swap Meet booth, (3) the RSSC logo design and possible contest to help get the best design (we're still trying to find a good use as a prize for our donated multimeter!), and (4) miscellaneous outstanding activities.

Several new members attended the meeting, so discussions also covered the Society's background, current activities, and future plans.

Jerry Burton presented the status of the Society's purchase of equipment from SynPet. We now have most of the major elements to our Society robot!

July 7th RSSC ROBOT PROJECT WORKSHOP

On July 7th, the RSSC had another robot project workshop at *The Robot Company* shop in Costa Mesa with about 10 members present.

We discussed the Society's purchase of a high performance controller (HPC) board, motor drivers, dc-dc converter, the TI speech board, and a Polaroid sensor harness for the RSSC robot. We still need some elements, such as a hard and floppy disk drives and controller, video board, and monitor. Several members at the workshop volunteered to donate many of the needed items. We now need to start putting it all together!

Our next workshop will be held on August 11th at 10:00 a.m., again at *The Robot Company*, 881 West 18th Street, Costa Mesa. Come join us at the workshop and see what's taking shape with our robot!

THE ROBOTEER

(Editor's note: Jerry Burton has again answered my plea for contributed material with another fine article.)

Last month I defined the concept of a 'Roboteer' and why I call my robot 'Dumbot.' This month I will begin to outline the steps that need to be taken to create a 'Smartbot.'

When asked what a robot can do, rather than get into a lengthy explanation,

I would like to able to merely give the verbal command, "Dumbot, get me a beer!" and have it go do it, regardless of where it is in the house. If I could have a robot that could do that, it would truly be a 'Smartbot.' I don't know if the person who asked what the robot could do would be impressed, but I certainly would be.

My dog can't get me a beer, and he is considered intelligent. He obeys my commands, although not very often. He recognizes his name when I call him and he will come to me. I have taught him to sit up for a biscuit and he remembers how to do it. He can get quite creative in figuring out how to get into a closed trash can to retrieve something his sensors tell him is in there.

What makes my dog smart and my robot dumb? One major difference is that my dog can sense his environment and change his behavior based upon changing conditions. Another is that he can learn simple behaviors and respond to my commands. He can also exhibit independent behavior that may or may not be acceptable (e.g., barking when someone or something comes within his range of awareness). Certain dogs, when properly programmed (trained), exhibit highly intelligent behavior and actually become partners to their human masters (e.g., seeing-eye dogs, police dogs, sheep dogs). It seems that this partnership approach yields the greatest benefit of robotics.

What capabilities must 'Dumbot' have in order to become a faithful servant or partner?

From a hardware viewpoint, it must be mobile, be able to recognize my voice, execute my verbal commands, have at least one arm to fetch things (or as a minimum, a basket to carry things, ala the *TOPO* approach), and must have sensors to be able to interact with its environment.

Mobility in this sense implies that the robot can move autonomously, which, in turn, requires that it be able to sense its environment and adjust to unforeseen circumstances.

It follows that the first step toward creating this kind of intelligent machine is make it 'aware' of its environment by providing it input via hardware and providing programmed adaptive behavior that allows it to interact 'intelligently' with that environment. It needs to know what the environment is like and where it is at any time.

When given the command, "Get me a beer," what must the robot do to perform that command.

First we need a top-level command processor than can coordinate what lower-level functions are to be performed and in what sequence. This command processor is implemented in Newton as a program called the Knowledge_Base. The Knowledge_Base is an interpreter that can execute C-like commands to perform various tasks. The software group is currently investigating replacing this program with a Forth-based command language, because Forth is already an interpretive language itself new 'commands' can be added quite easily.

This top-level command processor gets its input from either the keyboard, remote controller, or speech input subsystem. In any case the processor merely acts as 'traffic cop' in directing what programs are to be executed in a particular order.

The robot first has to recognize my verbal commands. No problem - the TI recognition system can convert my voice commands into ASCII commands that are placed in the keyboard buffer. The command 'Dumbot' gets its attention. The command "Get me a beer" invokes a series of commands in the Knowledge-base (e.g., GOTO KITCHEN, GET_BEER, RETURN.

Note that we break the initial command in to subgoals or objectives. The original command breaks down in to (1) move from where you are to where the beer is, (2) perform the task GET_BEER, and (3) return to the original location with the beer. If you wanted to get really sophisticated, the final subgoal could be GOTO ME, which implies the robot has some means of locating 'you.' For this discussion, I assume that the robot will

ely bring the beer back to the original carting location.

So now we have taken the original goal and broken it in to three subgoals. We need a system architecture that allows the subgoals to be met, which in turn solves the original problem.

For the moment, let's forget about the GET_BEER task and concentrate on the movement subsystem that needs to be provided so that the first and third subgoals are achievable.

Figure 1 shows an overall system architecture that divides the general movement problem into a series of modules that perform the functions required to determine what the environment is like and respond to top-level commands to move through the environment.

The knowledge base (or some equivalent high-level command processor) would first invoke the Planning module with the command GOTO KITCHEN LOCATION BEER.

The Planning module translates GOTO KITCHEN to a location BEER within the kitchen class. The planning module must first compute a path from the robot's current location to the task location, then feed these points into the Navigation module for execution. Once the robot is positioned at the final location, it returns to the high-level command processor, which then executes the task process that was associated with the 'Beer' (e.g., a program called GET_BEER). Once GET_BEER has completed its task, it will return to the high-level command processor with a code indicating whether it was successful or not. The command processor then executes the Planning module with a RETURN command, and Planning then calls the Navigation module with the original points in reverse order to finish the top-level command 'Get..Beer' by RETURNing to the starting position.

Note that the planning module is very general (i.e., it merely has to determine how to get from where it is to where a task is to be performed and back). Therefore, the planning module can be used to perform an unlimited number of 'tasks' as long as it knows 'where' the tasks are to performed.

Planning relies on the Navigation module to carry out its requests. Planning is not the least bit concerned with 'how' Navigation is done or what mode of locomotion is used (e.g., wheels, tracks, legs). Navigation will return to the Planning module with its current position and a result code indicating success or failure in achieving the desired path. Planning's only function is to determine whether the requested LOCATION is known to the robot and providing a series of X,Y points to get the robot there (and back, if requested). If Navigation was not successful, then Planning returns to the command processor with a request to ask the user for help and abort the command.

Navigation is responsible for moving from point to point and avoiding local obstacles it encounters (if avoidance is on). It may or may not rely on a global environment map (E-map) (more on this when I cover the Navigation module in more detail in a subsequent article).

The Mapping module is responsible for creating a global E-map for use by the Planning and Environment_Definition module. In a subsequent article, I will go through the various methods currently in use to achieve this goal.

The Environment_Definition module is where the user tells the robot what tasks are defined and 'where' they are to be performed. The user interacts with the robot via a display and keyboard. This module also allows the user to modify or augment the robot generated E-map and to make decisions concerning 'what' things are. The user can also provide nontask specific locations, landmarks, and beacon locations (more on this later).

The Manual module merely provides the nonvoice or 'direct control' interface between the user and the Navigation module. It should also allow a learning mode so that the user can 'program' the robot directly. This module merely takes the commands received by the manual input(e.g.,direct connect joystick, teach

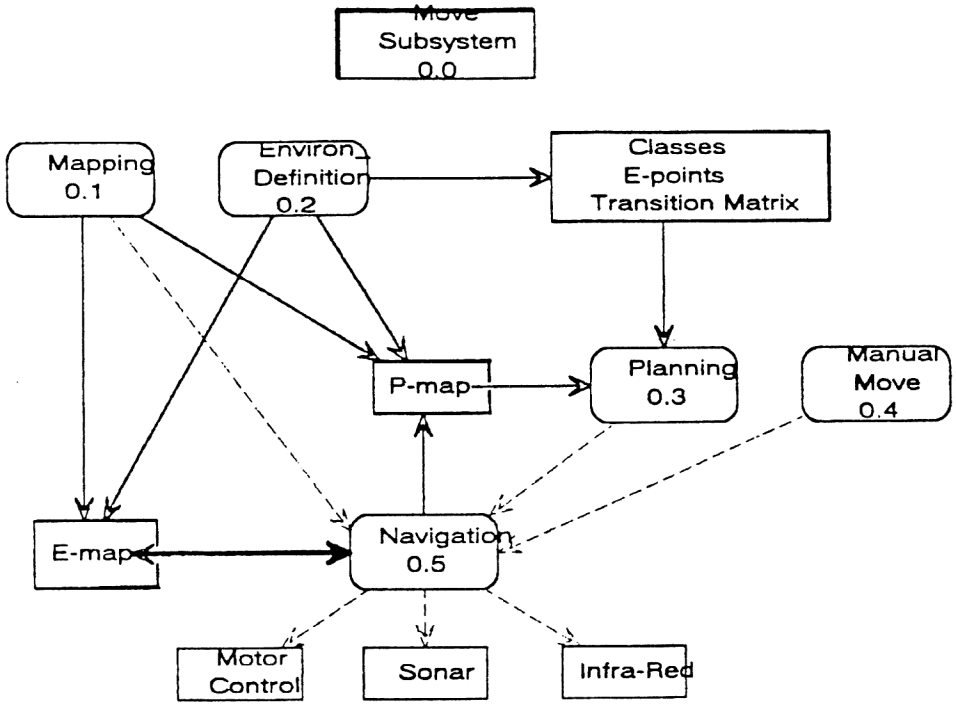


Figure 1 - Movement Subsystem Overall Architecture

pendant, RF receiver, IR controller) and converts them into X,Y points for input to the Navigation module.

Next month I will provide a more detailed look at some of the strategies that have been used (with references for those of you that like to read academic papers) and detail the approach I am taking in each of the modules so far defined.

One final note. The overall system design should allow for a number of different strategies to be investigated, without having an effect on other modules in the system. This can be implemented using an object-oriented design approaches that tends to insulate portions of the implementation from other parts, as long as the interface between the modules remains the same.

For example, the Planning, Mapping, and Manual modules all provide a series of points to the Navigation module. The Navigation module is totally independent of how the points were determined. Therefore, we can experiment with many different planning methods as long as the interface remains the same.

Conversely, the Planning module does not expect any particular method to be used by the Navigation module, so we can experiment with many different navigation methods.

Because the modules do not depend on each other's methods, we gain a great deal of flexibility. They do, however, expect a consistent interface (i.e., a series of points in and a final position out).

Oops, got to go, Dumbot's calling me for his next lesson. Next month, I'll go into more detail on the Planning module.

Jerry Burton

COMING RSSC EVENTS

For our August 7th meeting, we are still making arrangements to have a company representative provide a presentation. It is tentatively planned to go back to the ACP swap meet on July 29th to continue attracting new members and individuals interested in robotics. We hope to have the Society's robot ready to show and in addition have more of our members' robot projects.

I am planning to update the RSSC Resource Directory in August. Anyone who has information on a listing that should be included, contact me at the August 7th meeting or send it to the RSSC mailing address.

JULY 3RD MEETING AGENDA

1) Business agenda
   - RSSC Logo designs
   - OC Fair and ACP booth results
2) Presentation of RSSC robot design and construction status
3) Presentation - Subject still TBD
4) RAM (Random Access Meeting) - bring something of interest to share with the membership!

Again, I hope to see the entire membership there, along with any interested individuals or business representatives! Pass the word around about the RSSC!!

Scott MacGillivray, Editor

The following was taken from the *Olympic Robot Building Manual* prepared by the M. Artificial Intelligence Laboratory, and may be of interest to the membership.

The Futaba Gyro

Paul Viola

The Futaba gyro is a one axis rate gyro. It contains a single gyro whose rotation axis is perpendicular to the intended rotation of the chassis. As the chassis is rotated, the gyro experiences a torque that is dependent on the rate of rotation. Since the gyro is spring mounted, its deflection angle is linearly related to that torque. The deflection angle is measured and is interpreted as the rate of rotation of the chassis.

Pulse Width Encoding

The gyro assembly is controlled by a small box packed full of complex analog and digital hardware. This gyro assembly needs to be driven with a pulse width modulated signal in order to run. In return, it emits a pulse width modulated signal signifying the rate of turn. That is, the Futaba gyro controller takes a pulse encoded value and modifies it (increasing or decreasing the pulse width) based on the current rate of rotation.

The Futaba standard pulse width encoded output signal is a 20 msec cycle where 1500 micro seconds ON is the center of the range that can be measured (i.e. a 50 hertz square wave with a 7.5 percent duty cycle). A smaller ON period (lower duty cycle) implies smaller pulse width encoded values while a larger ON period (or higher duty cycle) implies a larger pulse encoded value (the range is approximately 1000-2000 micro secs).

Interface to the 6811

To interface a 6811 to the gyro controller amplifier, the 6811 needs to provide a 1.5ms wide pulse train tuned to 50 hertz. This provides the base from which the rate of rotation is measured. The output is then another square wave at 50 hertz with a modified pulse width. This width corresponds to the rate of rotation.

The gyro has two inputs and one output. The input of interest is labeled RX 1-4 on the data sheet. It is normally connected to a radio controlled receiver for a model airplane. This should be connected to your 50 hertz 1.5ms pulse generator (one of the pins on your 6811 which you've programmed). The output, labeled SX, should be connected to one of the 6811's timer input capture lines.

On each of the connectors, the output signal is white, ground is black and red is power.

Snip off the external battery power lines (unless you have a 6 volt battery power supply around). You will power the the gyro through the red and black power lines on the RX 1-4 input port. The 6811 has a sophisticated set of timers and a timer output compare register can be used to generate the base 1.5ms signal. This should be a straightforward use of some simple interrupts. Similarly, the timer input capture register can be used to measure the width of the incoming pulses; another simple interrupt application. Both of these facilities are described in the 6811 data sheet's section on timers.

Adjustments

The documentation included with the gyro describes some adjustments such as gain, center and reverse. The gain adjustment controls the amount of amplification. Center controls the

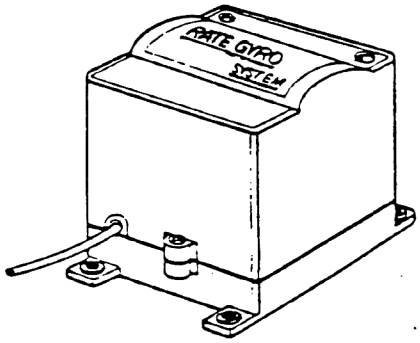
neutral or center of the pulse width encoding (what I have been assuming is 1.5ms). Reverse inverts the effect on the pulse width of a particular rotation (this is more important when used in a model aircraft - you could do the same by changing a sign somewhere in your program).

Warnings

Readings are not especially reliable. The device is a *rate* gyro; it is useless for determining angular position over even a small period of time. Even when stationary, the gyro will "drift" and additionally, there is a hysteresis effect.

Futaba®

DIGITAL PROPORTIONAL
RADIO CONTROL



INSTRUCTION MANUAL

SINGLE AXIS RATE GYRO

FP-G152 (For J,M, and SG Series)
FP-G132 (For E, F, G, H, and L Series)

FUTABA CORPORATION OF AMERICA
FUTABA CORPORATION

D60226

Thank you for buying a Futaba digital proportional radio control set.
Please read this manual carefully before using your new set.

FUTABA

FUTABA CORPORATION OF AMERICA
555 West Victoria Street, Compton, Calif. 90220, U.S.A.
Phone: 213-537-9610, Telex: 23-0691227 Facsimile: 213-637-8529

FUTABA CORPORATION
Tokyo Office: Daido Bldg., 3-1-16, Sotokanda, Chiyoda-ku, Tokyo, Japan.
Phone: (03) 255-5881 Telex: J26532

Printed in Japan/851120

The FP-G132/G152 is a single axis rate gyro designed to stabilize aircrafts. Like full size aircrafts, stabilisation is accomplished by detecting angular accelaration with the rate gyro. Detected motion information is fed to the control amplifier, which then sends a counteraction signal to the appropriate control surface.

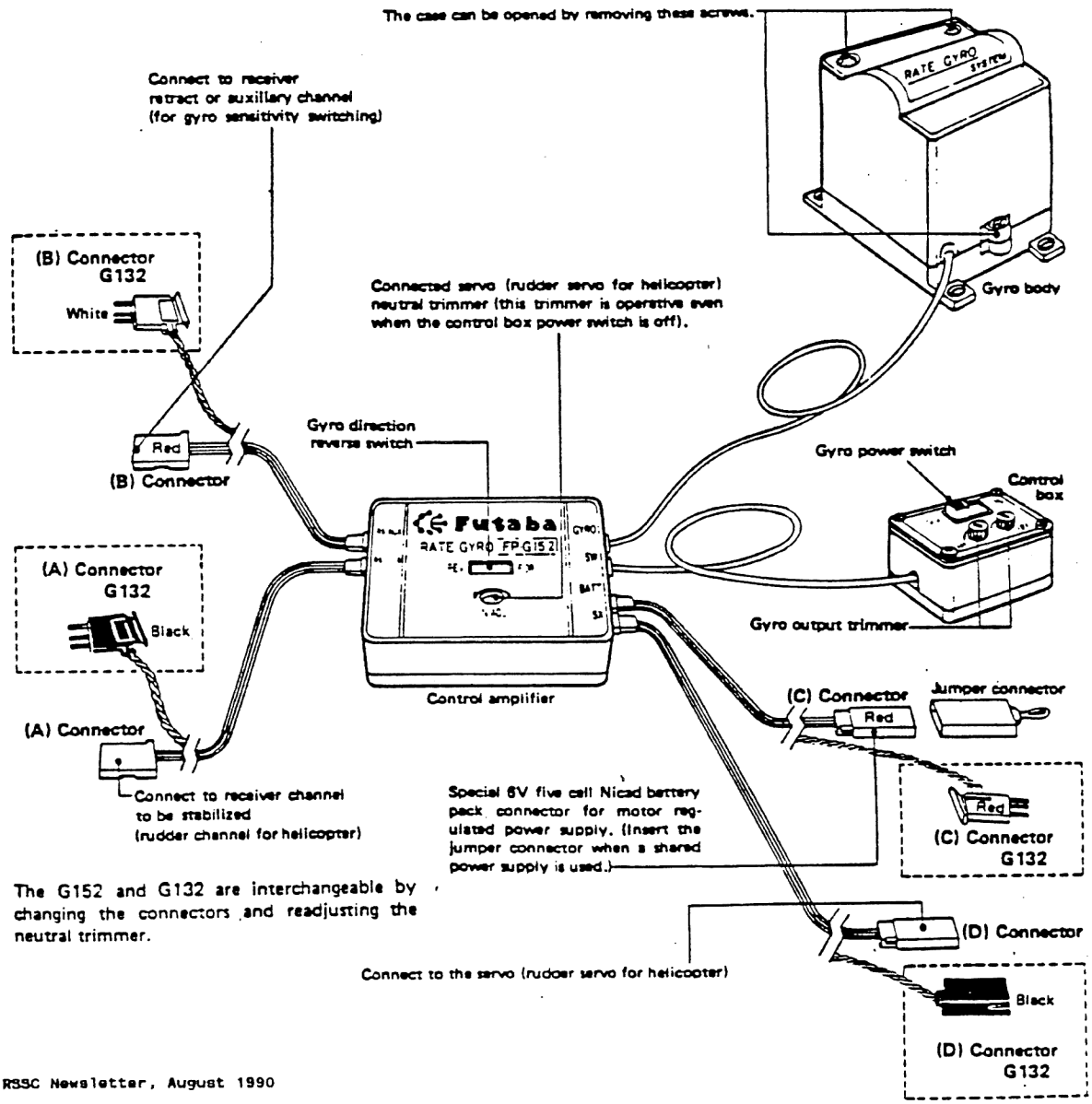
FEATURES OF FP-G152
FEATURES OF FP-G132

- The FP-G152 is for Futaba J, M, and SG Series (1520 μs, neutral) digital proportional radio control sets.
- The FP-G132 is for Futaba E, F, G, H, and L Series digital proportional radio control sets.
- Voltage regulated gyro motor supply maintains constant motor speed and allows consistent gyro performance. The voltage regulator is effective only when used with an external 6V. (five cell Nicad battery).
- A very sensitive magnetic motion sensor with excellent voltage characteristics, linear sensitivity, high speed response is used. This results in superior neutral characteristics. Such characteristics make it ideal for use in the rudder channel of a model hilicopter or in the aileron/elevator channel of a model aircraft.
- Large 2mm diameter gyro motor shaft for long life and strength.
- The gyro can be bypassed without affecting normal operation by turning the gyro power switch "off".
- Mount the gyro with grommets and screws or use two-sided foam tape.
- Gyro output ("sensitivity") can be switched to one of two preset outputs at the transmitter (use the retract switch if possible, or any avail. channels).
- Direction of correcting mix can be switched at the control amplifier (internal reverse amp switch).
- Centering of the channel being stabilized can be adjusted by a neutral trimmer built into the control amplifier.

INGS

Power supply voltage	4.8V, shared with receiver (6V for external supply)
Current drain	Motor: 100mA, Amplifier: 20mA (at 4.8V)
Dimensions and weight	Gyro body: 1.57 x 1.65 x 1.69 in.(40 x 42 x 43mm) - 2.86 oz.(80g)
	Control amplifier: 1.73 x 2.28 x .63 in.(44 x 58 x 16mm) - 1.61 oz.(45g)
	Control box: .94 x 1.34 x .59 in.(24 x 34 x 15mm) - .54 oz.(15g)

CONNECTIONS



The G152 and G132 are interchangeable by changing the connectors and readjusting the neutral trimmer.