Trikebot Java API Reference v2.1

by Andrés Santiago Pérez-Bergquist <aspb@mapache.org>

Most of the following functions are methods of the TrikebotController class. Assuming your TrikebotController is called `trikebot`, invoke them with commands of the form `trikebot.RefreshState()`

Unless otherwise specified, all functions return a boolean: true = success, false = failure See also the error code field of the state.

Networking Commands

InitComm(int trikebotPort, int cameraPort)

Open serial connection directly to robot; not currently implemented.

InitComm(String ipaddr)

Open TCP/IP connection to iPaq.

CloseComm()

Close any open communication channels.

Fundamental Commands

RefreshState()

Refreshes the state variables. All functions update the status, and all motion functions update the entire state (range-finder reading, speed, etc.). This is only for use if you haven't commanded the robot in a while and want to update the state.

InitRobot()

Reset to default mode and kill any programmable behaviors.

Motion Commands

SetDriveTimeout (int time)

Sets the delay between giving the robot a command to set the wheel velocity and having the robot stop, specified in seconds. Default is 5 seconds.

SetVel (int vel)

Sets the wheel velocity. Accepts a number from -100 (full reverse) to 100 (full forward). (As with all values, this can be calibrated for a specific robot.) Using this is generally not recommended, as it is likely that the steering servo will be powered down while driving by this method, allowing the wheel to turn freely under the stresses of driving.

TurnWheelTo (int turn)

Sets the wheel angle in degrees. Zero is straight ahead, negative is left, positive is right. Range is about -90 to +90.

Drive (int vel, int turn)

Simultaneously sets velocity and wheel angle. (Note that the wheel may take some time to turn to the desired angle, but the velocity starts ramping up immediately. The wheel can take up to four seconds to turn, so for big changes in

direction, you may with to use SetTurnAngle, Sleep, and then SetVel.)

PanHeadTo (int pan)

Sets the horizontal camera angle in degrees. Zero is straight ahead, negative is left, positive is right. Range is about -90 to +90.

TiltHeadTo (int tilt)

Sets the vertical camera angle in degrees. Zero is straight ahead, negative is down, positive is up. Range is about -45 to +90.

MoveHeadTo (int pan, int tilt)

Simultaneously sets pan and tilt angles.

SetParameters(boolean tmset, int timeout, boolean vset, int vel, boolean tset, int turn, boolean cpset, int pan, boolean ctset, int tilt)

Set the timeout for the motor, the velocity, the heading, and the pan and tilt angles, with a mask for each. (If a mask value is true, set the corresponding parameter, otherwise ignore it.) These are in human-sensible values, as described above. See also calibration commands.

KillMotor()

Kills all power to the wheel motor (coasts briefly instead of braking).

Calibration Commands

The calibration of the robot is controlled by the publicly accessible constants in CalibrationConstants. The various constants ending in MIN and MAX define the range of raw servo or motor speed values that will be outputted. The corresponding constants ending in _ANGLE or _SPEED give the human-readable values that these will map to linearly. (Note that is possible for the human-readable value that a MIN maps to be greater than the value MAX maps to; this has the effect of reversing the servo).

The best way to deal with these is to just use the calibration panel provided in the GUI and then move around the code files it generates for you when you're done.

Camera Commands

These are all methods of the CMUcam class. To invoke them, use calls of the form `trikebot.camera.ResetCamera()`

Note: The following are always set

Tracking Light = auto mode

Middle Mass Mode = on

Noise Filtering = on

Switching Mode = off

ResetCamera()

Returns us to the default state: RGB mode, auto white balance on, autogain on, mass mode enabled, poll mode enabled (IMPORTANT), noise filter on, full window size

SetContrast(int contrast)

Set the contrast to a value between 0 and 255.

SetBrightness(int brightness)

Set the brightness to a value between 0 and 255.

SetAutoWhiteBalance(boolean)

Turn Auto White Balance on (true) or off (false).

SetAutoGain(boolean)

Turn Auto Gain on (true) or off (false).

SetWindow(int x1, int y1, int x2, int y2)

Set the window for the next operation

GetVersion()

Returns a String with the firmware version.

GetMean()

Get the mean color of the current window. The values are stored as publicly readable int-valued variables in `trikebot.camera.mean` The means and standard deviations of the red, green and blue values are stored in the fields `rMean`, `gMean`, `bMean`, `rDev`, `gDev`, and `bDev`. That is, to access the mean of the red pixel values, use `trikebot.camera.mean.rMean`

GrabFrame()

GrabFrame(int winx1, int winy1, int winx2, int winy2)

Starts separate thread to grab a screenshot, which will be displayed in the GUI. With no arguments, grabs the entire window. With arguments, it only grabs that portion of the window. Note that either version overwrites any previously set window.

GrabFrameAndWait()

GrabFrameAndWait(int winx1, int winy1, int winx2, int winy2)

Like GrabFrame, but the call does not return until the image has been fully downloaded to the computer (or an error has occured). This should take about 5 to 7 seconds.

WaitForGrabFrame()

Returns once the current frame grab has been fully downloaded to the computer (when the currently executing frame-grabbing thread terminates). If no frame grab is currently taking place, it returns immediately.

Image()

Returns an Image object with the most recently taken picture.

WaitForImage()

Like Image(), except that it will not return until the current frame grab completes, ensuring that you get a full image (assuming no error occurred). If no frame grab is currently taking place it just returns the most recently taken picture.

GrabFrameAndKeep(String title)

Performs a frame grab and then copies the resulting Image to a new ImageWindow with the title given. Does not return until the frame grab is complete.

TrackColor(int rmin, int rmax, int gmin, int gmax, int bmin, int bmax)

Track the specified color once and return the results in `trikebot.track` (of limited utility; see also ActiveColorTracking)

TrackWindow()

Track the dominant color in the current window once and return the results in `trikebot.track` Additionally, the mean of the initial scene is returned in `trikebot.camera.mean` (of limited utility; see also ActiveWindowTracking)

At some point in the future, line mode will be enabled to allow visual feedback while tracking.

Complex commands

ActiveColorTracking(int rmin, int rmax, int gmin, int gmax, int bmin, int bmax)

Trains the camera on the largest visible object whose color lies in the the specified range and then automatically adjusts the pan and tilt servos to keep the camera centered on the object.

ActiveWindowTracking()

Waits five seconds for the camera to adjust to local lighting conditions, then trains the camera on the largest visible object currently in view at that time and automatically adjusts the pan and tilt servos to keep the camera centered on that object.

StopActiveTracking()

Cancels previous following modes. Auto white balance and auto gain will now be off.

ActiveColorTrackingNoTilt(int rmin, int rmax, int gmin, int gmax, int bmin, int bmax)

ActiveColorTrackingNoPan(int rmin, int rmax, int gmin, int gmax, int bmin, int bmax)

ActiveColorTrackingNoMove(int rmin, int rmax, int gmin, int gmax, int bmin, int bmax)

ActiveWindowTrackingNoTilt()

ActiveWindowTrackingNoPan()

ActiveWindowTrackingNoMove()

Like the original versions, except that iPaq sends no tracking-induced motion commands to the tilt servo, the pan servo, or both. You can still command that servo from the Java client.

int[] Sweep(int start, int end, int step)

Sweeps the range-finder from the start to the end position and returns an array of readings every step units (can bring up graphic display on laptop). Only works if the range-finder is mouted on the rotatable drive wheel assembly.

Safety(int reading)

Set the iPaq to automatically brake if an obstacle is too close, as determined by a range-finder larger than the given threshold. The iPaq currently ignores this.

ResetComputedPosition (boolean resetX, boolean resetY, boolean resetTheta)

Reset that part of the dead reckoner. (Of no use until the dead reckoner is implemented.)

After dead reckoning gets implemented, there will exist commands to calibrate how the integration takes place (set the relationship between velocity and position).

Framegrab Manipulation Commands

These are all methods of the CamCanvas class, and are invoked by calls of the form `trikebot.camera.pic.ColorAt(x, y)`

Image()

Returns the currently displayed Image object, just like `trikebot.camera.Image()`

Pixels()

Returns an array (`int[80*143]`) each of whose values represents one pixel in the current image, coded in RGB format. The `java.awt.Color` class is very useful for decoding such values and reconstructing them. They are ordered row-major from the top left, as is usual. To access pixel (x, y) in logical coordinates (where the bottom left is (1,1) and the top right is (80, 143), look at array element `[80*(143-y)+x-1]`.

SetPixels(int[] newPixels)

Accepts an array containing a new image that completely replaces the existing image. The image must be encoded as above. If the array is not 80 * 143, bad things will happen.

ColorAt(int x, int y)

Returns a Color object containing the color of the pixel at location (x, y), where $1 \le x \le 80$ and $1 \le y \le 143$.

RedAt(int x, int y)
GreenAt(int x, int y)
BlueAt(int x, int y)

Returns an int containing the red, green, or blue component of the color of the pixel at location (x, y), where $1 \le x \le 80$ and $1 \le y \le 143$.

Configure BrainStem

These are methods of class BrainStem, and are invoked by calls of the form `trikebot.brainstem.SetP(20)`

SetP(int P)
SetI(int I)
SetD(int D)
SetPeriod(int period)
SetLatency(int latency)

If you need to talk to the BrainStem directly, use the `BrainStemDialogue(byte[])` method of TrikebotController, which sends the specified bytes directly to the BrainStem.

Macros

These are methods of the Macro class, and are invoked by calls of the form
`trikebot.macros.StartRecording()`
Unless otherwise stated, these methods return a boolean where true indicates success and false indicates failure.

Defined()
: Returns a String[] holding the name of every currently defined macro. (The array is sorted by ASCII value, so upper-case letters will precede lower-case ones.)

StartRecording()
: Starts logging all commands sent to the robot for future replaying. StartRecording should not be called again until either StopRecording or CancelRecording have been called once. Commands that require additional processing (DumpFrame and ActiveTracking) should not be used.

StopRecording(String macroName)
: Stops logging and stores any data recorded since StartRecording in a macro with the given name. If such a macro already exists, it will be overwritten. StopRecording should not be called unless StartRecording has been called and neither StopRecording nor CancelRecording have been called since.

CancelRecording()
: Stops logging commands and disposes of any data recorded since StartRecording. CancelRecording should not be called unless StartRecording has been called and neither StopRecording nor CancelRecording have been called since.

PlayBack(String macroName)
: Sends the commands contained in the named macro to the robot. Commands that require additional processing (DumpFrame and ActiveTracking) will not work.

Open(String macroName, File file)

Open(String macroName, String fileName)
: Opens the file specified either by the File object or the pathname and stores the contained macro under the name given.

Save(String macroName, File file)

Save(String macroName, String fileName)
: Saves the macro with the given name in the file specified either by the File object or the pathname.

Timing-related Commands

These are static methods of the Time class, invoked by calls of the form
`Time.Current()`

Current()
: Returns the current time in milliseconds as a long.

Sleep(int msecs)
: Do nothing for msecs milliseconds.

State information

The state information is stored in `trikebot.state` as read-only variables available through accessors which return int, via calls of the form `trikebot.state.Status()`

Status()

The status code returned by the last command.

0	Success (anything else is an error, indicating no action was taken)
1	Incorrect Checksum
2	Unknown Packet Type
10	Camera timed out
11	Camera spewed out too many response characters
20	BrainStem failed to respond

Range()

Most recent range-finder reading, as a value from 0 to 255. Note that larger values equal closer readings

Pan()

Tilt()

WheelAngle()

The pan, tilt, and wheel angles when being updated by some process, such as ActiveTracking. (These are the same degree readings that you hand when setting them.)

WheelVelocity()

The raw voltage rating returned by the BrainStem. (This may be made more user-friendly.)

WheelCurrent()

The raw amperage rating returned by the BrainStem. (This may be made more user-friendly.)

IntegratedX()

IntegratedY()

IntegratedTH()

The current estimated position of the robot based on dead-reckoning. Not yet implemented, so they're always zero.

Information returned by the iPaq when performing tracking is stored in an instance of TrackInfo in publicly accessible int-valued variables that can be read by commands of the form `trikebot.track.conf`

x	The position of the blob
y	
width	The size of the blob
height	
pixels	
conf	The camera's confidence

Useful functions in UserThread

Inside UserThread, CameraClick is called whenever the user clicks on the image in the Framegrab Window if any previous instances of CameraClick have stopped running. The x and y passed to CameraClick are the position of the pixel clicked on; x is an integer from 1 to 80, and y is an integer from 1 to 143. The r, g, and b passed are the color of the pixel clicked on. WaitForCameraClick does not return until the user clicks on the image again. The x and y values of the most recently clicked pixel can be obtained with X() and Y(), and the color of the most recently clicked pixed can be obtained with R(), G(), and B().

Example:

```
CameraClick(int x, int y, int r, int g, int b)
{
        WaitForCameraClick();
        DoSomethingWithRectangle(x, y, X(), Y());
}
```

Most programs that you write will implement behaviors that last an indefinite time. For your convenience, a function called Loop() is provided. Loop() returns `true` until either another button has been clicked on, or you set the variable `quit` to `false` (by issuing the line "`quit = false;`" in your code). By depending on Loop(), you can make sure that you do not have runaway threads that refuse to quit. A typical behavior function is shown below:

```
private void TypicalBehaviorFunction()
{
        // Set up anything before the control loop
        while (Loop())
        {
                // Main loop
        }
        // Do any needed cleanup
}
```

To end a function that checks Loop(), simply have a button that calls a function like this:

```
private void Stop()
{
        trikebot.KillMotor();
}
```

The very act of clicking a button to call Stop() will break the other function out of its while loop, and the KillMotor command ensures that the robot is not still driving. (Functions should stop driving and tracking as part of their cleanup, but this is just to make sure.)