Towards a Dataflow Approach to Robot Programming

Orestis Melkonian

Software & Knowledge Engineering Laboratory (SKEL)

NCSR "Demokritos"

Overview

Robot Programming

- **Common Patterns**
  - Robot perception architecture
  - Feedback loop controllers
- **ROS**: Robot Operating System
  - Hardware abstraction
  - Reusability
  - Language-agnostic open-source middleware
  - Publish-Subscribe design pattern
  - Communication via "topics"
  - Status Quo
    - Almost all code written in C++ and Python
    - Callbacks
    - Dataflow nature so far ignored

Stream Framework

- Topics as streams
- At a micro-level, replace callback "internal plumbing" with clean functional declarations
- At a macro-level, acts as a coordinating language adding to the composability of ROS
- **Extensibility**
  - Strategy design pattern for evaluation
  - Coder simply declares a dataflow graph
- **Advantages**
  - Decouple design (what to do) from execution (how to do it)
  - Cleaner, easier to maintain code
  - Implicit concurrency
  - Implicit message-passing

Future Work: Extensions

- User-defined operators
- Legacy code as internal dataflow nodes
- "Well-behaved" existing nodes mix with newly created ones

Future Work: Optimization

- General operator reordering
- Network-aware adaptation

Future Work: DSL

- Avoid verbosity
  - Specific annotations (operator properties, execution directives)
- Restrict host language
  - Single-assignment
  - Only pure functions in second-order operators
- Impose a specific program structure
  - Minimize design flaws

The End

Embed laser in camera video

```
Topic LASER = new Topic("/scan", LaserScan._TYPE);
Topic CAMERA = new Topic("/camera/rgb/image_color", Image._TYPE);

Stream.setEvaluationStrategy(new RosEvaluationStrategy());

// ROS Topics
Stream<LaserScan> laser = Stream.from(LASER).filter(LaserScan::valid);
Stream<Mat> camera = Stream.from(CAMERA).map(OpenCv::convertToMat);

// Combine
Stream.combineLatest(laser, camera, (l, im) -> {
    int width = mat.width(), height = mat.height();
    Point center = new Point(width / 2, height);
    float curAngle = l.getAngleMin();
    for (float r : l.getRanges()) {
        double x = (center.x + (width / 2 * r * Math.cos(curAngle + Math.PI / 2)));
        double y = (center.y - (width / l.getRangeMax() * r * Math.sin(curAngle + Math.PI / 2)));
        if (Math.abs(curAngle) < 0.3)
            Core.line(mat, center, new Point(x, y), new Scalar(0, 0, 255));
        curAngle += l.getAngleIncrement();
    }
    Core.circle(mat, center, 2, new Scalar(0, 0, 0), -1);
    return im;
}).subscribe(window::show);
```

Surveillance

```
Topic CAMERA = new Topic("/camera/rgb/image_color", Image._TYPE);

Stream<Mat> image = Stream.<Image>from(CAMERA).map(OpenCV::convertToMat);
Stream<Mat> initial = image.take(1).repeat();
Stream.zip(image, initial, Pair::new)
      // stop monitoring when video stream stops
      .timeout(1, TimeUnit.MINUTES)
      .doOnCompleted(this::shutdown)
      // Only stream images when motion is detected
      .filter(Surveillance::motionDetected)
      .map(Pair::getLeft)
      .subscribe(window::showImage);
```

Control Panel I

```
// Sensors
Topic LASER = new Topic("/scan", LaserScan._TYPE);
Topic CAMERA = new Topic("/camera/rgb/image_color", Image._TYPE);
Topic DEPTH = new Topic("/camera/depth/image", Image._TYPE);
Topic TF = new Topic("/tf", TFMessage._TYPE);
// Actuators
Topic CMD = new Topic("/cmd_vel", VelCmd._TYPE);

Stream.setEvaluationStrategy(new RxjavaEvaluationStrategy());

// ROS topics
Stream<Point> laser = Stream.from(LASER).filter(LaserScan::valid).map(LaserScan::getCartesianPoint);
Stream<Mat> image = Stream.<Image>from(CAMERA).map(OpenCV::convertToMat);
Stream<TFMessage> tf = Stream.from(TF);
Stream.<Image>from(DEPTH).map(OpenCV::convertToGrayscale).subscribe(viz::displayDepth);

// Embed laser in color image
Stream.combineLatest(laser, image, (l, im) -> return im.embed(l))
    // Detect faces
    .sample(100, TimeUnit.MILLISECONDS)
    .map(this::faceDetect)
    // Display
    .subscribe(viz::displayRGB);
```

Control Panel II

```
// TF frames
tf.take(50).collect(HashMap::new, (map, msg) -> {
    List<TransformStamped> transforms = msg.getTransforms();
    for (TransformStamped transform : transforms) {
        String parent = transform.getHeader().getFrameId();
        String child = transform.getChildFrameId();
        if (!map.containsKey(parent)) {
            Set<String> init = new HashSet<>();
            init.add(child);
            map.put(parent, init);
        } else map.get(parent).add(child);
    }
})
.subscribe(viz::displayTF);

// Battery
Stream.interval(2, TimeUnit.SECONDS).map(v -> (100 - v) / 100.0).subscribe(viz::displayBattery);

// Control
Stream.<KeyEvent>from(KEYBOARD).map(CmdVel::new).subscribe(CMD);
```

C++ I

```
bool scanReceived = FALSE, imageReceived = TRUE;
LaserScan scan; Image image;
subscribe<LaserScan>("scan", scanCallback);
subscribe<Image>("/camera/rgb/image_color", imageCallback);

while (ros::ok()) {
    if (scanReceived && imageReceived) {
      window.show(merge(scan, image)));
      scanReceived = FALSE;  imageReceived = FALSE;
    }
    ros::spinOnce();
}

void scanCallback(LaserScan newScan) {
  if (!scanReceived) {
    scan = newScan;
    scanReceived = TRUE;
  }
}

void imageCallback (Image newImage) {
  if (!imageReceived) {
    image = new Image(newImage);
    imageReceived = TRUE;
  }
}
```

C++ II

```
Mat merge_stream(LaserScan scan, Image image) {
  Mat mat = OpenCV.convertToMat(image);
  int width = mat.width(), height = mat.height();
    Point center = new Point(width / 2, height);
    float curAngle = l.getAngleMin();
    for (float r : l.getRanges()) {
        double x = (center.x + (width / 2 * r * Math.cos(curAngle + Math.PI / 2)));
        double y = (center.y - (width / l.getRangeMax() * r * Math.sin(curAngle + Math.PI / 2)));
        if (Math.abs(curAngle) < 0.3)
            Core.line(mat, center, new Point(x, y), new Scalar(0, 0, 255));
        curAngle += l.getAngleIncrement();
    }
    Core.circle(mat, center, 2, new Scalar(0, 0, 0), -1);
    return mat;
}
```

Hamming Numbers

```
Stream.just(1).loop(entry -> mergeSort(
            mergeSort(
                    entry.map(i -> 2 * i),
                    entry.map(i -> 3 * i)
                ),
            entry.map(i -> 5 * i)
        )
    )
    .startWith(1)
    .subscribe(System.out::println);

Stream<Integer> mergeSort(Stream<Integer> s1, Stream<Integer> s2) {
    Queue<Integer> queue = new PriorityQueue<>();
    return Stream.zip(s1, s2, (i1, i2) -> {
        Integer min = Math.min(i1, i2), max = Math.max(i1, i2);
        queue.add(max);
        if (min < queue.peek())
            return min;
        else {
            queue.add(min);
            return queue.poll();
        }
    }).concatWith(Stream.from(queue));
}
```