Appendix A: More Details on the Changepoint Detection Algorithm

This appendix provides more detail on the changepoint detection algorithm discussed in sections 2.3 and 3. This algorithm is based on a Hidden Markov Model, which is shown in Figure 7. The model q_t at each time t is hidden, but produces observable data y_t. Transitions occur when the model changes, either to a new model or the same model with different parameters. The transition from model q_i to q_j occurs with probability $g(j-i-1)p(q_j)$, while the emission probability for observed data $y_i, ..., y_{j-1}$ is $P(i, j, q_i)(1-G(j-i-1))$. These probabilities are considered for all times $i < j$ and models $q_i, q_j \in Q$.

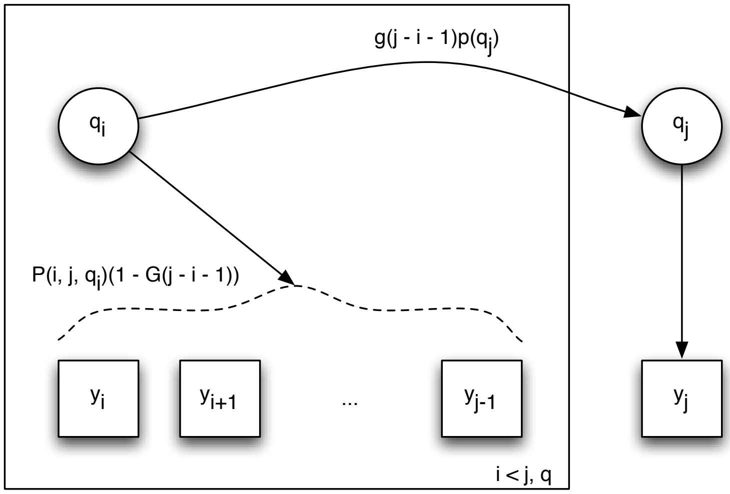

Figure 7: The Hidden Markov Model modeling changepoint detection.

Using this model, Fearnhead and Liu [8] use a Viterbi algorithm to compute the MAP changepoint positions and selected models, by processing each sample on the trajectory as it is experienced and thus without having to store the trajectory. Psuedo-code for this algorithm is given in Figure 8.

When using Fearnhead and Liu's changepoint detection algorithm in a reinforcement learning setting and using the models we have selected in Section 3, we must compute sufficient statistics for a regression over R (return) without storing the trajectory sample, even though we only receive r (reward) at each step. The sufficient statistics required to compute $P(j, t, q)$ are $\mathbf{A}$, $\mathbf{b}$ and $(\sum_{i=j}^{t} R_i \Phi(\mathbf{x}_i))$. An update rule that incrementally computes these sufficient statistics without storing the trajectory is given in Figure 9.

Appendix B: Acquiring Skills from Human Demonstration on the uBot: Implementation Details

The uBot was given six abstractions, each corresponding to the pairing of one of two types of motor abstraction with one of three colored markers. When a marker was paired with the arm endpoint, the abstraction's state variables consisted of real-valued difference between the endpoint and the marker centroid in 3 dimensions. Actions were real-valued vectors moving the endpoint in 3 dimensions. When a marker was paired with the robot's torso, the abstraction's state variables consisted of two real values representing the distance and angle to the marker centroid. Actions were real-valued vectors controlling the speed and direction of the robot torso using differential drive.

Particles were generated according to the currently executing motor abstraction, and a switch in motor abstraction always caused a changepoint.[3] The parameters used for performing CST on the uBot were $k = 50$, $M = 60$ and $N = 120$, using a 1st order Fourier basis.

When performing policy regression, we used ridge regression over a 5th order Fourier basis to directly map to continuous actions. We set the regularization parameter λ by 10-fold cross-validation, using a test set sampled from the same trajectories as those used to perform the fit. For testing, we varied the starting point of the robot by hand and used hard-coded stopping conditions that corresponded to the subsequent skill's initiation set.

---

[3] Informal experiments with removing this restriction did not seem to change the number or type of skills found but in some cases changed their starting and stopping positions by a few timesteps.

```
particles = $\emptyset$
Process each incoming data point
for t = 1:T do
    Compute fit probabilities for all particles
    for $p \in$ particles do
        p_tjq = (1 - G(t - p.pos - 1)) $\times$ p.fit_prob() $\times$ model_prior(p.model) $\times$ p.prev_MAP
        p.MAP = p_tjq $\times$ g(t - p.pos) / (1 - G(t - p.pos - 1))
    Filter if necessary
    if $|particles| >= N$ then
        particles = particle_filter(particles, particles.MAP, $M$)
    Determine the Viterbi path
    if t == 1 then
        max_path = []
        max_MAP = 1/$|Q|$
    else
        max_particle = $\max_{\mathrm{p}}$ p.MAP
        max_path = max_particle.path $\cup$ max_particle
        max_MAP = max_particle.MAP
    Create new particles for a changepoint at time t
    for $q \in Q$ do
        new_p = create_particle(model = q, pos = t, prev_MAP = max_MAP, path = max_path)
        particles = particles $\cup$ new_p
    Update all particles
    for $p \in$ particles do
        p.update_particle($\mathbf{x}_t$, $y_t$)
Return the most likely path to the final point.
return max_path
```

Figure 8: Fearnhead and Liu's online MAP changepoint detection algorithm.

```
input : $\mathbf{x}_t$, the current state; $r_t$, the current reward
Initialization
if t == 0 then
    $\mathbf{A}_q$ = zero_matrix(q.m, q.m)
    $\mathbf{b}_q$, $\mathbf{z}_q$ = zero_vector(q.m)
    sum_r$_q$, tr_1$_q$, tr_2$_q$ = 0;
Compute the basis function vector for the current state
$\Phi_t = \Phi_q(\mathbf{x_t})$
Update sufficient statistics
$\mathbf{A}_q = \mathbf{A}_q + \Phi_t \Phi_t^T$
$\mathbf{z}_q = \gamma \mathbf{z}_q + \Phi_t$
$\mathbf{b}_q = \mathbf{b}_q + r_t \mathbf{z}_q$
$\text{tr\_1}_q = 1 + \gamma^2 \text{ tr\_1}_q$
$\text{sum\_r}_q = \text{sum\_r}_q + r_t^2 \text{ tr\_1}_q + 2\gamma r_t \text{ tr\_2}_q$
$\text{tr\_2}_q = \gamma \text{ tr\_2}_q + r_t \text{tr\_1}_q$
```

Figure 9: Incrementally updating the changepoint detection sufficient statistics for model q.