

# Um quadrotoor totalmente autônomo em ambiente indoor

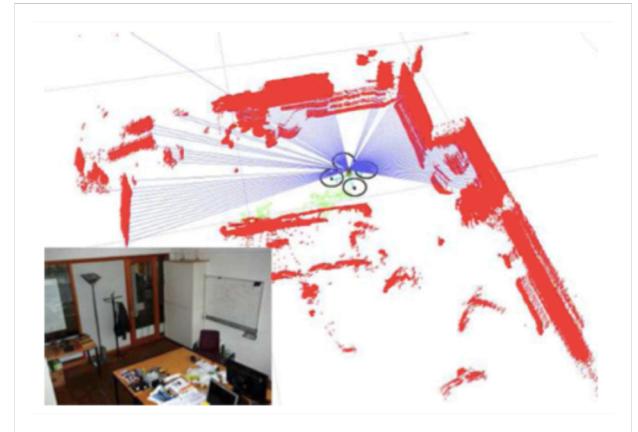
Slawomir Grzonka, Giorgio Grisetti e Wolfram Burgard

**Resumo**—Recentemente, o interesse no desenvolvimento de veículos voadores autônomos vem crescendo. No entanto, como a maior parte das abordagens propostas são adequadas para operação outdoor, apenas algumas técnicas foram projetadas para ambientes indoors, onde dispositivos não podem contar com o Sistema de Posicionamento Global (GPS) e, portanto, necessitam usar seus sensores exteroceptivos para navegação. Neste artigo, apresentamos um sistema de navegação genérico que permite a um sistema quadrotoor de pequeno porte operar de forma autônoma em ambientes indoors. Para atingir este objetivo, que sistematicamente consiste na extensão e adaptação de técnicas aplicadas com sucesso em robôs terrestres, nós descrevemos todos os algoritmos e apresentamos um amplo conjunto de experiências, que ilustram a possibilidade de navegação autônoma de um quadrotoor de forma confiável em ambientes indoors.

**Palavras-chave**—Navegação, quadrotoor, localização e mapeamento simultâneos (SLAM), veículos autônomos não tripulados (VANT)

Recentemente, a comunidade robótica tem aumentado o interesse em veículos aéreos autônomos, especialmente quadrotores. Plataformas de voo de baixo custo e de tamanho pequeno estão se tornando cada vez mais disponíveis e, algumas delas, são capazes de levantar cargas relativamente altas e proporcionar cada vez mais um amplo conjunto de funcionalidades básicas. Tal fato nos leva à possibilidade e pensar em como equipá-las com habilidades de navegação autônoma. No entanto, a maioria das abordagens propostas para voos autônomos [14], [32] foca em sistemas para operação outdoor. Já em ambientes indoors, os veículos autônomos têm um grande potencial de utilização em uma variedade de aplicações, que incluem vigilância, busca e salvamento [10]. Em tais condições e comparados com veículos terrestres, a principal vantagem de dispositivos voadores é a sua maior mobilidade.

Da mesma forma que veículos terrestres, a tarefa principal de um robô voador autônomo consiste em chegar a um local desejado de forma não supervisionada, ou seja, sem interação humana. Na literatura, esta tarefa é conhecida como navegação ou orientação. Para lidar com a tarefa geral de navegação é necessário enfrentar um conjunto de problemas que vai desde estimar a pose do dispositivo até o planejamento de sua trajetória. Vários são os sistemas disponíveis atualmente que tratam o problema de navegação de veículos terrestres em ambiente indoor e exterior de forma eficaz [1], [2].



**Fig. 1:** Um voo autônomo de nosso quadrotoor em um escritório repleto de obstáculos. O espaço livre ao redor do robô está seriamente limitado, impondo grandes exigências em estabilidade de pose, localização e controle. A imagem no canto inferior esquerdo mostra a sala de escritório a partir de um ponto de vista semelhante ao da foto.

No entanto, os princípios gerais de algoritmos de navegação que têm sido aplicados com sucesso em robôs terrestres, poderiam, em princípio, ser transferidos para veículos aéreos, mas essa transferência não é simples por várias razões. Robôs terrestres são inherentemente estáveis, no sentido de que, ao atribuir zero à sua velocidade, isso fará com que o robô desacelere suavemente até parar. O mesmo não se aplica para robôs voadores que precisam serativamente estabilizados, mesmo quando eles já estão no local desejado. Além disso, por causa da dinâmica rápida de um veículo voador em comparação com um robô terrestre, todas as quantidades necessárias para estabilizar o veículo deverão ser calculadas dentro de um curto espaço de tempo e com um nível adequado de precisão. Assim, a portabilidade de sistemas de navegação de robôs terrestres para veículos aéreos obriga a atender restrições mais rigorosas no que diz respeito à precisão e eficiência.

Neste artigo, apresentamos a tecnologia que permite a navegação autônoma de um quadrotoor em ambientes indoors e descrevemos um sistema de navegação que inclui funcionalidades-chave, ou seja, localização, planejamento, estimativa de superfície, aprendizagem do mapa e controle. No entanto, como um veículo voador se movimenta em 3-D no ambiente indoor, geralmente não há estrutura suficiente para descrever o ambiente com representações em 2-D. Ao invés de usar uma representação completa 3-D, baseamo-nos numa representação 2-D com as paredes representadas como elevação do piso. A vantagem desta escolha comparada com a representação 3-D é que se pode operar em uma grande variedade de ambientes indoors usando variantes de algoritmos

eficientes para 2-D que trabalham com representações densas de mapas, em vez de métodos 3-D considerados custosos para execução. A adaptação destas funcionalidades para o caso 3-D seria ou muito lento ou não precisa o suficiente dada a limitação de tempo para tornar o sistema estável. Este artigo estende nosso trabalho anterior [17], através da introdução de algoritmos de estimativa simultânea melhorada da altitude do veículo e da elevação da superfície base. Nós ainda fornecemos resultados quantitativos da nossa abordagem de mapeamento e localização simultâneos (SLAM) e discutimos o efeito de diferentes modos de correspondência de varredura incremental na estabilidade da pose do robô. Descrevemos também nossos algoritmos de planejamento de trajetória e desvio de obstáculos e fornecemos detalhes adicionais e experimentos realizados.

Nosso sistema é resultado da integração de hardware e software para atender as várias restrições desafiadoras impostas por veículos voadores de pequeno porte, preservando um elevado grau de flexibilidade. Ele ainda pode ser operado em diferentes níveis de autonomia, tais como: na ajuda ao piloto, impedindo colisões com obstáculos e mantendo a posição do veículo quando não são dadas ordens; na construção de mapas online durante o voo em um ambiente desconhecido; ou na instrução para chegar autonomamente em determinados locais através de um mapa conhecido. Nosso sistema foi avaliado em um quadroto de código aberto denominado Mikrokopter [3]. A **Fig. 1** ilustra o nosso sistema quadroto e seu estado, enquanto autonomamente voando dentro de uma sala de escritório repleto de obstáculos.

## I. TRABALHO RELACIONADO

Na última década, as plataformas de voo receberam um aumento atenção da comunidade científica. Muitos autores focaram na modelagem e no controle destes veículos [8], [11], [25], [29], com especial destaque para pequenos e micro helicópteros [10]. Hoffmann et al. [19] apresentou um algoritmo baseado em modelo para voos autônomos com seu quadroto STARMAC. Seu sistema faz voos outdoors, utilizando GPS e unidades de medida iniciais. Ng et al. [14] têm desenvolvido algoritmos para aprender a controlar a navegação de helicópteros de forma autônoma. A abordagem deles permite que os helicópteros realizem manobras impressionantes em ambientes ao ar livre. Scherer et al. [28] descreve algoritmos de voo ágil entre obstáculos a baixa altitude através da utilização de um scanner a laser. Tempelton et al. [30] demonstra como usar a visão para o mapeamento de terreno ao ar livre e aterrissagem autônoma. Tournier et al. [33] e Bourquardez et al. [12] usou visão para estimar e estabilizar a pose atual de um quadroto. Thrun et al. [32] usou um helicóptero controlado remotamente para aprender modelos 3-D em grande escala ao ar livre. Existiram também alguns trabalhos que abordaram a navegação de veículos voadores em ambientes indoors e na ausência do sinal de GPS. Nesse ambiente, vários autores utilizaram visão para controlar ou auxiliar o controle de um quadroto [7], [20], [21]. Roberts et al. [26] usou sensores de ultrassom para controlar um veículo voador em um ambiente de teste estruturado, enquanto He et al. [18] apresentou um sistema de navegação de um quadroto de pequeno porte sem a utilização de GPS. Aqui, a posição do veículo é estimada por um *unscented Kalman filter (UKF)*. Sempre que o robô tem que chegar a um determinado local, um

caminho que garante uma boa densidade de observação é calculado a partir de um mapa predefinido. Estas observações altamente densas minimizam o risco de falhas de localização. Em paralelo com o nosso trabalho, Achtelika et al. [6] desenvolveu um quadroto autônomo para ambiente indoor equipado com um scanner a laser e câmeras que permitem um sobrevoo autônomo em um ambiente com restrição. Recentemente, Celik et al. [13] apresentou o seu sistema de SLAM indoor usando uma câmera monocular e ultrassom. Nossos artigos é ortogonal a um trabalho recente de Bachrach et al. [9], onde os autores apresentam um sistema para a realização de exploração autônoma e aquisição de mapas de ambientes indoors. Eles estendem as ferramentas de navegação 2-D para robôs CARMEN [27], adicionando um filtro de partículas Rao-Blackwellized para SLAM e um algoritmo para sistemas baseados em exploração autônoma de fronteiras. No entanto, eles não fornecem a localização, otimização de mapa, desvio de obstáculos, ou SLAM multinível. Além disso, nós utilizamos um algoritmo SLAM mais robusto baseado em grafos que permite a otimização do mapa e estimar a altitude da superfície abaixo do robô. Isso permite que um quadroto, equipado com o nosso sistema, voe sobre superfícies cuja altura é constante em partes (em degraus).

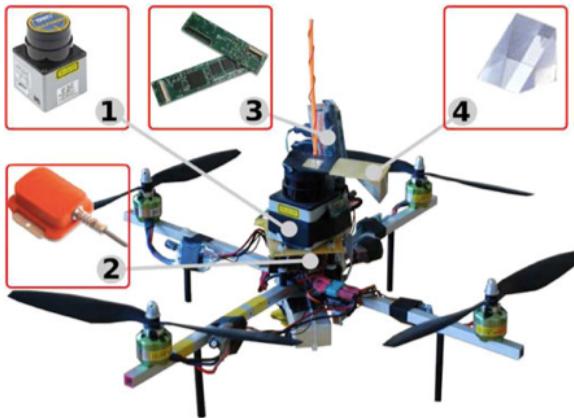
## II. A NAVEGAÇÃO EM UM VOO AUTÔNOMO DE UM QUADROTO EM AMBIENTE INDOOR

Para chegar a um local desejado de forma autônoma, um robô móvel tem de ser capaz de determinar um caminho livre de colisão da origem até o seu destino. Esta tarefa é conhecida como planeamento de trajetória e exige um mapa do ambiente a ser determinado. Normalmente, este mapa tem de ser adquirido pelo próprio robô através do processamento das medições dos sensores que são obtidas durante uma missão de exploração. Esta tarefa de geração do mapa é conhecida como mapeamento e localização simultâneos (SLAM). Para a maioria das aplicações, é suficiente realizar SLAM anteriormente através de uma sequência de gravação de medições. Para seguir o seu caminho, com uma precisão suficiente, o robô tem de estar ciente da sua posição no ambiente em qualquer momento. Esta tarefa é conhecida como a localização. Um outro componente fundamental de um sistema de navegação é o módulo de controle que tem por objetivo mover o veículo ao longo da trajetória, dada a posição estimada pela localização. Devido ao aumento do risco de danificar a plataforma durante o teste de voo, o piloto deve ter a possibilidade de assumir o controle da plataforma, em qualquer momento. Por fim, a dinâmica mais complexa de uma plataforma de voo impõe requisitos substancialmente mais rigorosos sobre a precisão do processo de estimativa de sua pose quando comparado com veículos terrestres típicos. Embora, em cenários ao ar livre, erros de precisão de posicionamento são aceitáveis em até um metro, isso não é possível em ambientes indoors, já que o espaço livre ao redor do robô é substancialmente mais reduzido.

## III. ARQUITETURA DE HARDWARE

A **Fig. 2** mostra um Mikrokopter [3], quadroto de código aberto equipado com sensores e dispositivos computacionais. O Mikrokopter vem com um controlador de baixo nível para a rolagem, arfagem e guinada. Nossos quadrotos são semelhantes ao

proposto por He et al. [18] e é composto pelos seguintes componentes: (1) um sensor laser miniatura Hokuyo-URG para SLAM e evitar obstáculos, (2) um XSens MTi-G, que são sistemas micro eletromecânicos IMU para estimar a atitude do veículo, (3) Gumstix, um PC integrado baseado em Linux com interfaces USB e uma placa de rede Wi-Fi, que comunica com o micro controlador no quadrotor através de uma interface RS-232 e (4) um espelho que é utilizado para refletir alguns dos feixes de laser ao longo do eixo z para medir a distância até o chão.



**Fig. 2:** Plataforma quadrotor que é usada para avaliar o sistema de navegação baseado num Mikrokopter e inclui (1) um sensor a laser Hokuyo, (2) XSens IMU, (3) computador Gumstix e (4) espelho de laser.

#### IV. SISTEMA DE NAVEGAÇÃO

Nosso sistema de navegação é baseado em uma arquitetura modular, em que diferentes módulos se comunicam através da rede, utilizando um mecanismo de publicação-assinatura. Em nossa plataforma atual, todos os drivers de dispositivo são executados embarcados, enquanto os algoritmos mais custosos computacionalmente são executados em um computador e se comunicam remotamente com a plataforma através de redes sem fio.

Uma vez que rolagem ( $\phi$ ) e arfagem ( $\theta$ ), medidos pela IMU são, em geral, precisos em até 1 grau, podemos usar diretamente esta informação no nosso sistema de navegação. Isso nos permite reduzir o problema de localização dos 6 graus de liberdade (6DOF), ou seja,  $(x, y, z, \phi, \theta, \psi)$  para 4DOF, com base na posição 3-D  $(x, y, z)$  e  $\psi$ , o ângulo de guinada. O único sensor que é usado para calcular estes 4DOF e detectar obstáculos é o medidor de distâncias à laser.

Com base em parâmetros conhecidos de calibração inicial e na altitude atual ( $\phi, \theta$ ) que são estimados pela IMU, projetamos o laser no sistema global de coordenadas. Dados os raios laser projetados, estimamos  $(x, y, z, \psi)$  do veículo em um mapa 2-D, contendo múltiplos níveis por célula. Para compensar a falta de medidas odométricas, estimamos os movimentos incrementais em  $(x, y, \psi)$  por varredura do laser 2-D. Finalmente, podemos controlar a altitude do veículo e estimar, simultaneamente, a elevação da superfície sob ele através da fusão do acelerômetro da IMU com a distância do solo medida pelo laser. Dessa forma, rastrear e mapear vários níveis dentro do ambiente, o que permite que o nosso robô mantenha corretamente sua

altura, mesmo quando ele voa por cima de obstáculos, tais como mesas ou cadeiras.

#### A. Estimação do Movimento Incremental

A varredura a laser mede, no instante  $t$ , um conjunto de distâncias  $r_t$  ao longo do plano  $xy$  em seu próprio referencial. Por isso, primeiro calculamos a projeção das distâncias medidas  $b_t$  para os feixes não refletidos pelo espelho utilizando a rolagem e arfagem estimados pela IMU. Consequentemente, calculam-se os pontos  $h_t$  para todos os feixes refletidos pelo espelho utilizando uma cadeia de transformações da IMU para a posição virtual do laser, que justifica o uso do espelho. Algumas tarefas, como estabilização, dependem da precisão da estimativa da pose local do veículo nos arredores. Para este fim, nós podemos estimar o movimento relativo do robô entre duas varreduras subsequentes, pelo uso de um algoritmo de correspondência de varredura. Uma vez que a altitude é conhecida a partir da IMU, este procedimento pode ser realizado em 2-D, assumindo ambientes indoors estruturados. Um algoritmo de correspondência de varredura estima a pose mais provável do veículo  $\hat{x}_t$  no instante  $t$  dadas  $k$  poses anteriores  $x_{t-k:t-1}$  e as medidas do laser correspondentes  $b_{t-k:t}$ , da seguinte forma:

$$\hat{x}_t = \underset{x:=(x,y,\psi)}{\operatorname{argmax}} p(x_t | x_{t-k:t-1}, b_{t-k:t}) \quad (1)$$

Para resolver (1), usamos uma variante da varredura da correspondência de multi-resolução correlativa proposta por Olson [24]. A ideia por trás de uma correspondência de varredura correlativa é discretizar o espaço de busca  $\mathbf{x}_t = (x_t, y_t, \psi_t)$  e realizar uma busca exaustiva nos parâmetros discretizados em torno de uma determinada estimativa inicial. Para avaliar eficazmente a similaridade  $p(x_t | x_{t-k:t-1}, b_{t-k:t})$  de uma dada solução  $\mathbf{x}_t$ , nós utilizamos campos de similaridade [31] obtido pelo mapa mais similar gerado através das últimas observações  $b_{t-k:t-1}$ .

A complexidade de uma correspondência de varredura correlativa depende linearmente da resolução na qual os parâmetros são discretizados e da faixa de pesquisa. Uma implementação simples deste algoritmo não é adequada para a nossa aplicação, que exige tanto de uma alta precisão e computação eficiente. Para superar esse problema, nós empregamos uma abordagem de multi-resolução. A ideia é realizar a pesquisa em diferentes resoluções: partindo de uma menor até uma maior. As soluções que se encontram em uma menor resolução são então usadas para restringir a pesquisa em uma maior resolução.

Na nossa implementação, usamos um modelo de velocidade constante para calcular a estimativa inicial para a pesquisa e executar a verificação de correspondência correlativa em três resoluções diferentes, (ou seja,  $4\text{ cm} \times 4\text{ cm} \times 0.4^\circ$ ,  $2\text{ cm} \times 2\text{ cm} \times 0.2^\circ$  e  $1\text{ cm} \times 1\text{ cm} \times 0.1^\circ$ ). Montamos a área de pesquisa  $r$  dependendo da velocidade máxima  $v_{max}$  do veículo e na frequência  $f$  do scanner como  $r = v_{max}/f$ .

Nós controlamos a posição do veículo, que é baseada nas velocidades que são estimadas pela correspondência de varredura. Deste modo, os desempenhos da verificação de correspondência desempenham um papel importante na estabilização do robô. Em particular, queremos ter uma

estimativa rápida, precisa e suave (ou seja, com menos oscilações). Para se ter uma ideia sobre a precisão desejada, considere um erro na estimativa de posição de  $\pm 2\text{cm}$ . Assumindo um sensor de frequência de 10 Hz, este erro leva a uma variação de 20 cm/s no cálculo da velocidade entre duas varreduras do laser. Isto, por sua vez, pode gerar comandos errados pelo controlador reduzindo a estabilidade.

Em nossa correspondência de varredura hierárquica, a estimativa de alta resolução é afetada por oscilações frequentes por causa da resolução limitada da área de busca da vizinhança. Embora estas oscilações podem, em geral, serem filtradas por um filtro passa-baixo, este tipo de filtragem poderia introduzir um desvio de fase no cálculo da pose e da velocidade (a pose que foi estimada num instante anterior). Para obter tanto uma estimativa precisa de posição quanto um sinal suave sem oscilações, vamos calcular a resposta como a média ponderada das estimativas de todas as varreduras de correspondência na hierarquia. Os pesos da soma caem sobre uma curva gaussiana que é centrada nas estimativas de maior resolução. Em diversas experiências, descobriu-se que a média ponderada das estimativas é melhor para o controle do que cada estimativa simples isoladamente, como mostrado na **Tabela 1**. A tabela contém os resultados experimentais que comparam os efeitos sobre a estabilidade de pose usando estimativas de varreduras individuais com a nossa abordagem de média ponderada. Todas as execuções refletem experiências onde o objetivo do quadroto erá pairar no mesmo local a 0,5 m de altura enquanto durou a bateria. Para avaliar quantitativamente a nossa abordagem, compararmos a média e o desvio padrão em ambos: posição e velocidade absoluta. Como pode ser visto, o uso de uma média ponderada das diferentes resoluções tem uma efeito positivo sobre o circuito de controle. Isto origina do fato de que a média ponderada tem um efeito de suavização da estimativa de pose mas não inclui qualquer mudança de fase para o sistema. Uma vez que é usado um modelo simplista do nosso quadroto (considerando a velocidade constante), usar a saída de média ponderada (com a previsão utilizada como uma estimativa inicial para a busca) é igual a execução de um filtro de Kalman que tem uma grande incerteza sobre a predição. A inclusão de um modelo mais sofisticado para a predição levaria a melhores estimativas, no entanto, o uso desta estratégia simplista foi suficiente para os nossos propósitos.

**Tabela 1:** Efeito do algoritmo de varredura na estabilidade da pose do robô

abordagem	4cm	2cm	1cm	média ponderada	unidade
<b>média(<math>x</math>)</b>	<b>0.107</b>	<b>0.105</b>	<b>0.149</b>	<b>0.066</b>	[m]
<b>média(<math>y</math>)</b>	<b>-0.045</b>	<b>0.060</b>	<b>-0.04</b>	<b>-0.05</b>	[m]
<b>std(<math>x</math>)</b>	<b>0.145</b>	<b>0.148</b>	<b>0.165</b>	<b>0.123</b>	[m]
<b>std(<math>y</math>)</b>	<b>0.081</b>	<b>0.088</b>	<b>0.087</b>	<b>0.076</b>	[m]
<b>média(<math> v_x </math>)</b>	<b>0.146</b>	<b>0.095</b>	<b>0.084</b>	<b>0.075</b>	[m/s]
<b>média(<math> v_y </math>)</b>	<b>0.159</b>	<b>0.106</b>	<b>0.09</b>	<b>0.072</b>	[m/s]
<b>std(<math> v_x </math>)</b>	<b>0.118</b>	<b>0.071</b>	<b>0.065</b>	<b>0.058</b>	[m/s]
<b>std(<math> v_y </math>)</b>	<b>0.117</b>	<b>0.083</b>	<b>0.072</b>	<b>0.057</b>	[m/s]

### B. Localização e SLAM

Se um mapa do ambiente é conhecido a priori, a localização pura (em contraste com SLAM) é suficiente para a estimativa dos 4DOF restantes do quadroto. Nós estimamos a posição 2-D ( $x, y, \psi$ ) do robô num determinado mapa por localização de

Monte Carlo [15]. A ideia consiste em utilizar um filtro de partículas para controlar a posição do robô. Aqui, amostramos a próxima geração de partículas, de acordo com o movimento relativo que foi estimado pela correspondência de varredura e avaliamos as partículas atuais utilizando campos de semelhança [31].

Nosso sistema pode adquirir modelos de ambientes desconhecidos durante os voos autônomos ou manuais através do mapeamento e localização simultânea do ambiente. O objetivo de um algoritmo SLAM é estimar tanto a posição do veículo quanto o mapa do ambiente através do processamento de uma sequência de medições obtidas durante movimentação no ambiente. Mesmo quando um mapa é conhecido a priori, um mapa local é necessário até que o robô seja localizado, quando rodando de forma autônoma. Em nosso sistema, nós usamos um algoritmo SLAM popular baseado em grafos. A ideia deste tipo de algoritmo é a construção de um grafo a partir de medições do veículo. Cada nó do grafo representa a posição do veículo no ambiente e uma medição tomada naquela posição. As medições (vértices do grafo) estão ligadas por restrições em pares que codificam as relações espaciais entre poses próximas do robô. Essas relações são determinadas pela busca por pares de medições obtidas em locais próximos. Sempre que o robô reentra numa região conhecida depois de andar por muito tempo em uma área desconhecida, os erros acumulados ao longo da trajetória se tornam evidentes. Esses erros são modelados por restrições conectando partes do ambiente que foram observadas durante intervalos de tempo distantes e são conhecidos, na comunidade SLAM, como fechamentos de laço. Para recuperar um mapa consistente, usamos um algoritmo de otimização de gradiente estocástico que localiza a posição dos nós que maximiza a semelhança das extremidades. A abordagem da otimização é discutida em detalhes em [16] e uma versão do código aberto está disponível em OpenSLAM [4].

Mais uma vez, nós restringimos o nosso problema de estimativa para 4DOF, uma vez que a altitude fornecida pela IMU é suficientemente precisa para os nossos propósitos de mapeamento. Além disso, assume-se que o veículo opera sobre uma superfície constante por partes e que o ambiente indoor é caracterizado por estruturas verticais, tais como paredes, portas, etc. Apesar de latas de lixo, ferramentas de escritório sobre uma mesa ou a mesa por si só estarem violando esta hipótese, a utilização de um mapa 2-D ainda é suficiente para mapeamento preciso e localização. Isso decorre do fato de que esses detalhes em geral só são visíveis em uma pequena porção da medição atual enquanto mapeando. Já uma escrivaninha, por exemplo, ajuda na localização, uma vez que existe uma clara diferença no eixo  $xy$  entre uma escrivaninha e uma parede próxima. Assim, nós restringimos nossa abordagem para estimar um mapa 2-D e uma trajetória 2-D do robô ao longo do 3DOF ( $x, y, \psi$ ), ou seja, mapeamos todos os objetos como se eles tivessem uma extensão infinita. A estimativa da trajetória é a projeção do movimento robô 6DOF sobre o plano do solo sobre o eixo  $z$ . Nós estimamos a altitude da plataforma, uma vez que a posição 2-D e a altitude são conhecidas, baseando-se no procedimento descrito na próxima seção.

### C. Estimação da Altitude

Estimar a altitude do veículo num ambiente indoor significa determinar a altura global em relação a um referencial fixo. Uma vez que o veículo pode se mover ao longo de um terreno não-plano, não podemos usar diretamente os feixes  $h$  refletidos pelo espelho. A nossa abordagem, portanto, simultaneamente estima a altitude do veículo e a elevação do solo sob o robô. No nosso processo de estimativa, assumimos que a posição  $(x, y, \psi)$  do robô no ambiente é conhecido a partir do módulo de SLAM descrito anteriormente. Nós assumimos que, além disso, a elevação da superfície sob o robô é constante por partes. Chamamos de "nível" cada uma destas regiões de superfície ligadas, com altitude constante. A extensão de cada nível é representada por um conjunto das células de uma grid 2-D compartilhando a mesma altura.

Uma vez que nosso sistema carece de sensores de altitude globais, tais como barômetros ou GPS para determinar a altitude do veículo, verificamos a altura do veículo em relação ao solo e mapeamos diferentes elevações por um sistema de dois estágios de filtros de Kalman. O Algoritmo 1 descreve a nossa abordagem de uma forma abstrata.

Na primeira fase, o filtro de Kalman é usado para controlar a altitude  $z$  e a velocidade vertical  $v_z$  do veículo por meio da combinação das medições de inércia, medições de altitude e os níveis já mapeados sob o robô. Na segunda fase, um conjunto de filtros de Kalman é usado para calcular a elevação dos níveis medidos atualmente pelo robô. Para evitar desvios no cálculo da elevação, nós atualizamos a altitude de um nível somente quando o robô mede aquele nível pela primeira vez ou quando o robô retorna nele (isto é, quando entra ou sai daquele nível particular).

Em detalhe, a primeiro filtro de Kalman estima o estado de altura  $z = (z, v_z)$  e a incerteza correspondente  $\Sigma_z$ . Primeiramente, prevemos a altitude atual e velocidade ( $\hat{z}_t$ ), dada a estimativa anterior,  $z_{t-1}, \Sigma_{z_{t-1}}$ , e a aceleração medida pelo IMU (veja a linha 4 do Algoritmo 1).

Os feixes refletidos pelo espelho podem medir mais do que um nível simultaneamente. Por exemplo, quando voando sobre uma mesa, pode acontecer que uma fração dos feixes serem totalmente refletidos pela mesa, alguns feixes são parcialmente refletidos pela mesa e parcialmente pelo chão, enquanto os feixes restantes são totalmente refletidos pelo chão. Por isso, busca-se, na vizinhança local do mapa-multinível atual, todos os níveis que poderiam ter gerado uma das altitudes medidas  $h \in h_t$  (assumindo a altitude do robô como  $\hat{z}_t$ ). Este passo é indicado na linha 5.

Se encontramos pelo menos uma correspondência, então usamo-la para calcular uma medição virtual para o chão (veja a linha 7). Nós usamos os níveis encontrados, a partir do mapa atual e os feixes correspondentes, para calcular uma única medição. Em outras palavras, calcula-se a medida que obteríamos se não houvesse obstáculos presentes sob o robô e utilizamos essas informações para a atualização da medição do filtro de Kalman, como mostrado na linha 8.

---

### Algorithm 1 Multilevel-SLAM

---

```

Input: beams deflected by mirror at time  $t$ :  $h_t$ 
Input: previous multilevel map:  $\hat{M}$ 
Input: elapsed time:  $\Delta t$ 
Input: current pose:  $x_t = (x_t, y_t)$  // output of SLAM module
Input: previous height state  $z_{t-1} = (z_{t-1}, v_{z_{t-1}})$ 
Input: previous height state uncertainty  $\Sigma_{z_{t-1}}$ 
Input: z-acceleration and uncertainty:  $a_z, \sigma_z$  // from IMU
Output: current height state:  $z_t, \Sigma_{z_t}$ 
Output: current multilevel map:  $M$ 

1: function Multilevel-SLAM
2: // ————— 1st stage: update height estimate —————
3: // KF is short for Kalman Filter
4:  $(\hat{z}_t, \hat{\Sigma}_{z_t}) = \text{KF}(z_{t-1}, \Sigma_{z_{t-1}}).\text{predictionStep}(\Delta t, a_z, \sigma_z)$ 
5:  $E = \hat{M}.\text{at}(x_t \pm \Delta x).\text{getExistingLevelsMatching}(h_t, \hat{z}_t)$ 
6: if  $E \neq \emptyset$  then
7:    $(\tilde{m}, \tilde{\sigma}_m) = \text{createVirtualHeightMeasurement}(h_t, E)$ 
8:    $(z_t, \Sigma_{z_t}) = \text{KF}(\hat{z}_t, \hat{\Sigma}_{z_t}).\text{measurementUpdate}(\tilde{m}, \tilde{\sigma}_m)$ 
9: else
10:    $(z_t, \Sigma_{z_t}) = (\hat{z}_t, \hat{\Sigma}_{z_t})$ 
11: end if
12: // ————— 2nd stage: update map —————
13:  $L = \text{estimateLevels}(h_t, z_t)$ 
14:  $M = \hat{M}.\text{addNewLevels}(L, x_t)$ 
15:  $M = M.\text{updateExistingLevels}(L, x_t)$ 
16:  $M = M.\text{extendExistingLevels}(L, x_t)$ 
17:  $M = M.\text{searchForLoopClosures}(x_t)$ 
18: return  $z_t, \Sigma_{z_t}, M$ 
19: end function

```

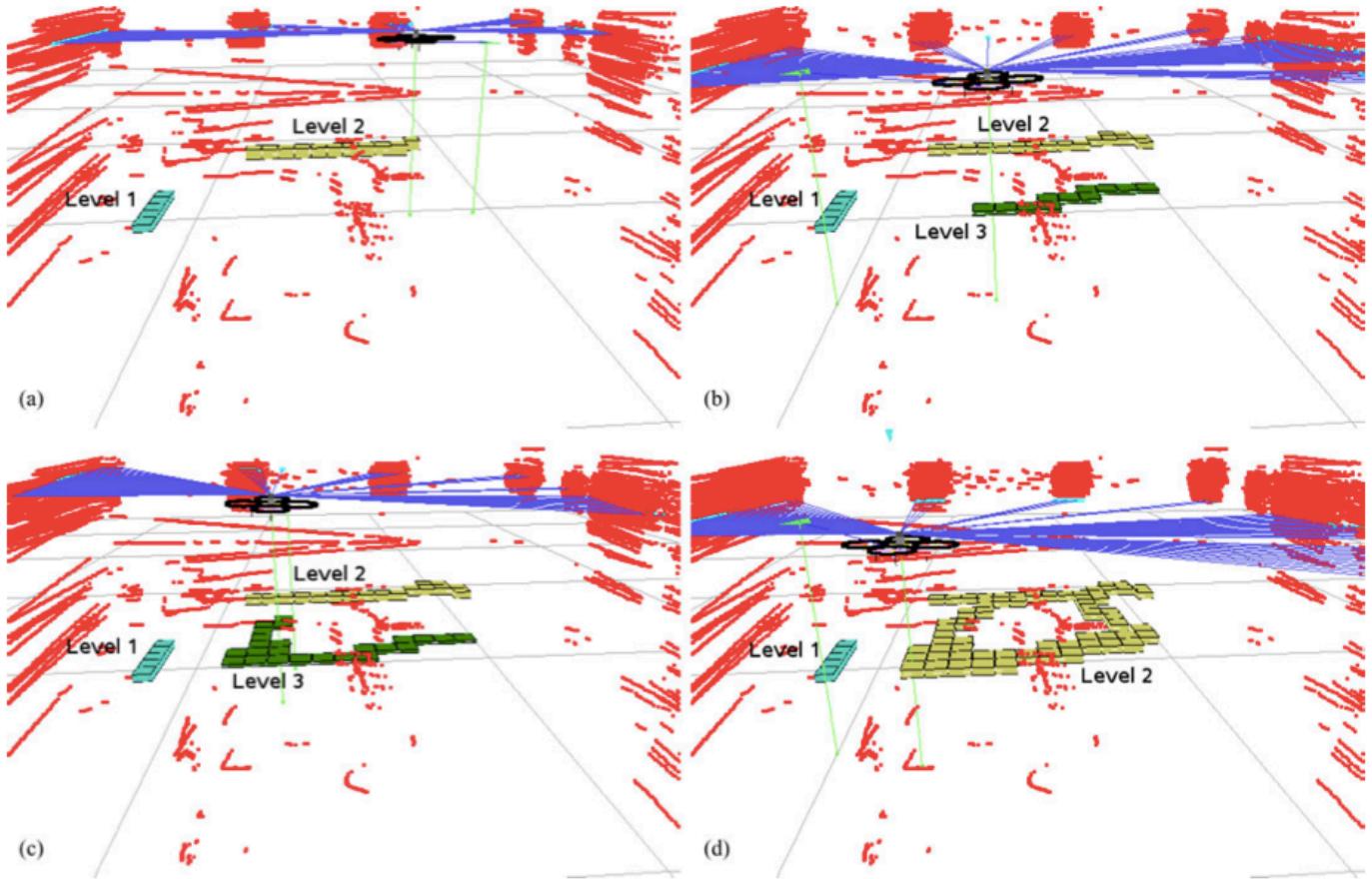
---

Algoritmo 1

No entanto, quando o robô explora o ambiente, pode acontecer que nenhum dos feixes atuais (ou seja,  $h \in h_t$ ) caia em uma região de confiança de um nível no mapa atual, ou seja,  $E = \emptyset$ . Neste caso, não podemos criar uma medição virtual e, portanto, somos incapazes de realizar a atualização da medição do filtro. Portanto, a previsão é então a melhor estimativa de altitude do robô, conforme descrito na linha 10.

Dada a altitude estimada do robô, agora podemos atualizar o mapa multinível atual. Lembre-se que os feixes refletidos pelo espelho podem medir mais do que um nível simultaneamente. Por isso, reunimo-os em conjuntos vizinhos. Cada um destes conjuntos é assumido para originar um único nível e é parametrizado pela média e a matriz de covariância calculada pelos feixes do conjunto. O resultado deste processo é o conjunto  $L$  que consiste nos níveis estimados como indicado na linha 13.

Assumimos que as medidas, que não caíram na vizinhança local da região de confiança dos níveis existentes, terem sido geradas por um novo nível de chão. Estes novos níveis de chão podem ser diretamente incluídos no mapa como mostrado na linha 14, no Algoritmo 1. Para todas as medidas que caíram na região de confiança de um nível no mapa, existem duas possibilidades: ou este nível já foi visto em um momento anterior, ou seja, o robô está voando sobre a mesa, e, portanto, ele já viu o nível correspondente antes, ou ele está entrando ou saindo deste nível particular. Neste último caso, podemos usar a estimativa de altitude atual, a fim de atualizar a altitude do



**Fig. 3:** Exemplo de junção de níveis durante a estimação de altitude do veículo e da elevação do solo abaixo dele. Cada nível é representado como um conjunto de células na grid 2-D que compartilham a mesma elevação. (a) O robô começa a explorar o escritório. Inicialmente, ele reconhece 2 níveis (Nível 1 e Nível 2) correspondendo à cadeira e mesa. (b) Após, ele voa para longe da mesa, retorna e voa sobre uma região diferente da mesma mesa. (c) Isso resulta na criação do Nível 3. Então, o robô mantém seu sobrevoo acima da mesa até que ele consiga estender o Nível 2 que tem o mesmo nível do que o Nível 3, o qual foi originado da mesma mesa. Esta situação é mostrada em (c). Finalmente, o robô entra no Nível 2 a partir do Nível 3. Nosso sistema reconhece que os dois níveis têm a mesma elevação, faz a junção e atualiza a elevação comum

nível do mapa (linha 15). A elevação de cada nível é rastreado por um filtro de Kalman individual.

Visto que armazenamos explicitamente objetos 2-D como uma extensão em  $xy$  ao invés de níveis individuais por célula, buscamos esses níveis presentes na vizinhança do mapa que são explicados por uma das medidas que foram obtidas. Se um nível for encontrado e não estiver presente na localização atual, estendemos este nível para a célula atual, como mostrado na linha 16.

Note que o robô observa apenas uma porção limitada da superfície de base. Assim, pode também acontecer que o robô "une" as superfícies de diferentes níveis de modo a formar uma nova. A **Fig. 3** ilustra essa situação. Inicialmente dois níveis correspondentes a uma cadeira (Nível 1) e uma mesa (Nível 2) são identificados (a). O robô então deixou a mesa para trás, faz uma curva, e voa sobre uma área diferente da mesma mesa. Uma vez que Nível 2 não está mapeado na vizinhança da pose atual, nosso sistema cria um novo nível (para a mesma mesa), que é nomeada como Nível 3 em (b). Por fim, o quadrotor continua a sobrevoar a área originalmente coberta da mesa que introduz uma interseção entre o nível 3 (nível atual) e o nível 2 (gerado anteriormente). Como consequência, ele junta os níveis 2 e 3 [ver **Fig. 3** (c) e (d)].

Quando dois níveis,  $L'_j$  e  $L'_k$ , com altitudes  $h_j$  e  $h_k$  e covariâncias  $\sigma_j^2$  e  $\sigma_k^2$  são mesclados, a estimativa de Gauss  $\langle h, \sigma^2 \rangle$  do nível combinado tem os seguintes valores:

$$\langle h = \frac{\sigma_k^2 h_j + \sigma_j^2 h_k}{\sigma_j^2 + \sigma_k^2}, \sigma^2 = \frac{\sigma_j^2 \sigma_k^2}{\sigma_j^2 + \sigma_k^2} \rangle \quad (2)$$

Esta etapa é indicada na linha 17 do Algoritmo 1.

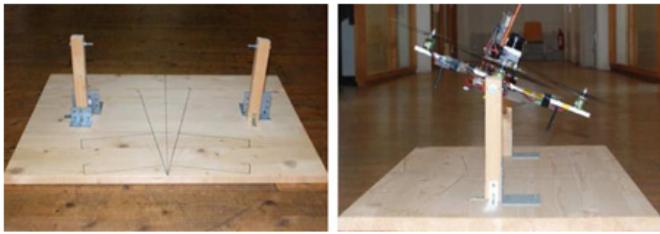
Para resumir, nós armazenamos um nível como um conjunto de células da grid 2-D representando a área de abrangência do objeto correspondente. Primeiramente, estimamos a altura atual do robô dado os níveis conhecidos no mapa multinível. Numa segunda etapa, atualizamos o mapa, dada a altitude estimada do robô. Aqui, um nível é constantemente re-estimado sempre que o veículo entra ou sai desse nível específico, e a associação de dados é resolvida pela posição conhecida ( $x, y, \psi$ ) do veículo. Finalmente, medições não explicadas por qualquer nível que se encontra presente no mapa são assumidas por terem sido geradas por novos níveis que são então incluídos no mapa.

#### D. Controle de alto nível para a pose e altitude

O algoritmo de controle de alto nível é usado para manter o veículo na posição atual. As saídas do algoritmo de controle são as variações na rolagem, arfagem, guinada e empuxo, que são denotados, respectivamente, como  $u_\phi, u_\theta, u_\psi$  e  $u_z$ . As

entradas são a posição e as estimativas da velocidade provenientes da correspondência de varredura incremental. Uma variação da rolagem se traduz num movimento ao longo do eixo  $y$ . Já uma variação nos resultados da arfagem, em um movimento ao longo do eixo  $x$ . E, por fim, uma variação da guinada resulta em uma mudança na velocidade vertical. Nós controlamos separadamente as variáveis individuais, por meio do controlador proporcional integral derivativo (PID) ou por controladores diferenciais proporcionais (PD). Uma vez que, no nosso caso, todos os comandos de controle são dependentes da estimativa da pose atual, nosso módulo de controle de alto nível é executado numa frequência de 10 Hz, já que o scanner a laser fornece medições nesta taxa.

Note que o Mikrokopter (e a maioria das plataformas comerciais disponíveis) vem com controladores de baixo nível para rolagem, arfagem, e guinada; Assim, não temos que tomar cuidado com o controle dos motores individuais, mas sim com o controle dos comandos, resultando em um ângulo desejado. No nosso caso particular, o controlador de baixo nível do quadrotor Mikrokopter roda a 500 Hz. Uma vez que comandos de guinada, em plataformas comuns, resultam em quanto rápido o quadrotor deve rodar e não quanto longe virar, esses parâmetros refletem a agressividade preterida do movimento de rotação na guinada. Ao contrário disso, os comandos de rolagem e arfagem resultam em ângulos desejados para os quais funções de mapeamento independentes devem ser aprendidas. A fim de entender esse mapeamento para nosso quadrotor, nós fixamos um eixo do veículo através de um suporte externo que permite o veículo rodar apenas ao longo do outro eixo. Nós aprendemos a função de mapeamento através do monitoramento do ângulo corrente medido pela IMU comparado com o comando enviado. Nossa plataforma para o teste de aprendizado desse mapeamento é mostrado na Fig. 4.



**Fig. 4:** Nossa plataforma de teste para aprender o mapeamento entre o comando e o ângulo correspondente. Este simples dispositivo permite a fixação de um eixo do quadrotor e monitorando o outro usando a IMU

Os comandos calculados são enviados diretamente para o micro controlador via RS232 que está no comando do controle de baixo nível (rolagem, arfagem e guinada) da plataforma. Por razões de segurança, o utilizador pode sempre controlar o veículo por meio de um controle remoto e nosso sistema mistura comandos do programa com os comandos do usuário. Durante nossos experimentos, nós permitimos que os programas perturbem os comandos do usuário por  $\pm 20\%$ . Dessa forma, se um dos módulos de controle falhar o usuário ainda tem a possibilidade de pousar com segurança o veículo sem qualquer perda de tempo já que ele não precisa pressionar qualquer botão primeiro.

Em particular, nós controlamos a arfagem e a rolagem através de dois PIDs independentes que são alimentados com as coordenadas  $x$  e  $y$  da pose do robô. A função de controle, em  $x$ , é a seguinte:

$$u_\phi = K_p \cdot (x - x^*) + K_i \cdot e_x + K_d \cdot v_x \quad (3)$$

Aqui  $x$  e  $x^*$  são a posição  $x$  medida e a posição  $x$  desejada, respectivamente.  $v_x$  é a velocidade correspondente, e  $e_x$  denota o erro integrado ao longo do tempo. O controle em  $y$  é análogo ao controle em  $x$ . Note que a parte integral poderia ser omitida (ou seja,  $K_i = 0$ ), mas nós encontramos uma melhor pairagem do veículo se um  $K_i$  pequeno é usado. Isto origina do fato de que, no nosso caso, somente os valores inteiros podem ser transmitidos para o micro controlador, embora o comando desejado seja um valor flutuante.

Nós controlamos a guinada pelo seguinte controle proporcional:

$$u_\psi = K_p \cdot (\psi - \psi^*) \quad (4)$$

Aqui  $\psi$  e  $\psi^*$  são a guinada medida e guinada desejada e  $u_\psi$  é a entrada do controle. A altitude é controlada por um controlador PID que utiliza a altura atual  $z$  estimada, a velocidade  $v_z$  e a corrente de tensão de bateria  $U_t$ , respectivamente. O controle  $u_z$  é definido como:

$$u_z = C(U_t) + K_p \cdot (z - z^*) + K_i \cdot e_z + K_d \cdot v_z \quad (5)$$

com  $K_p$ ,  $K_i$  e  $K_d$  sendo as constantes para o P, I e D e  $C(U_t)$  sendo uma constante de impulso, dada a corrente de tensão de bateria  $U_t$ . Aqui,  $z^*$  denota a altura desejada e  $e_z$  denota o erro integrado. Incluir um deslocamento de comando de impulso  $C(U_t)$  nos permite tratar o sistema como estacionário e, portanto, usar coeficientes constantes para o PID. Aprendemos  $C(U_t)$ , monitorando o impulso e o nível de bateria do veículo utilizando uma expectativa de maximização. Começamos com um controle PID, sem o  $C(U_t)$  e computamos o comando médio de impulso necessário para manter a altitude atual empiricamente através de vários voos de teste. Para cada nível de bateria  $U_t$  nós computamos o comando médio de impulso necessário para manter a altitude atual. Em voos subsequentes, usamos esse deslocamento como uma estimativa inicial para o  $C(U_t)$  e repetimos as experiências que resultaram em um refinamento para o  $C(U_t)$  até nenhuma grande mudança na estimativa acontecer.

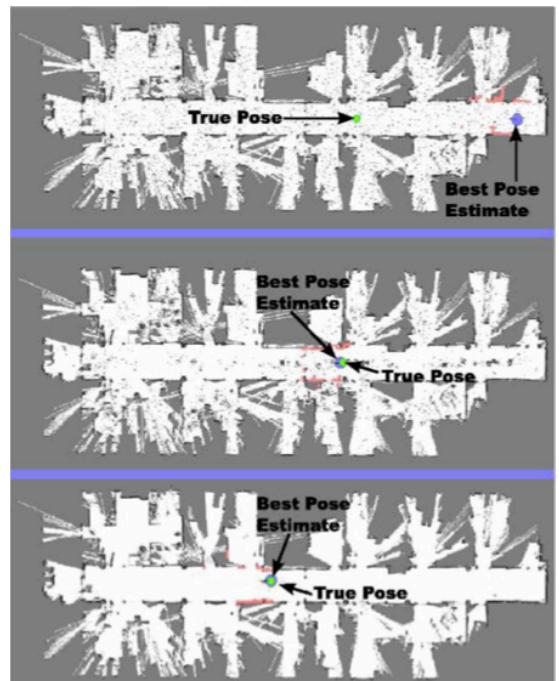
#### E. Planejamento de caminho e desvio de obstáculo

O objetivo do módulo de planejamento de trajetória é calcular um caminho, a partir do local atual, para um local objetivo definido pelo usuário, que satisfaz um ou mais critérios ótimos e é seguro o suficiente para evitar colisões, mesmo no caso de pequenas perturbações. A segurança é geralmente assegurada por escolher um caminho que seja suficientemente distante dos obstáculos no mapa. Finalmente, por causa do aumento do DOF de um veículo voador em comparação com um robô terrestre, o caminho deve ser planejado no espaço 4DOF ao invés de 3DOF. Em nosso sistema, usamos o algoritmo D\* Lite [22], que é uma variante do algoritmo A\* que pode reutilizar soluções anteriores para corrigir um plano inválido ao invés de recalculá-lo a partir do zero. Uma vez que o planejamento diretamente no 4DOF é

muito caro para o nosso sistema. nós computamos o caminho em duas etapas consecutivas. Primeiro, usamos D\* Lite para computar um caminho no espaço  $xyz$ , mas apenas consideramos as ações que movem o robô no espaço 2-D  $xy$ . Para cada localização  $(x,y)$ , sabemos, a partir do mapa multinível, a elevação da superfície que fica embaixo do robô. Esta elevação conhecida é usada para determinar uma possível mudança na altitude que o robô teria que tomar quando se desloca para uma célula próxima. Uma mudança na altitude é refletida pelo aumento nos custos para atravessar que é proporcional ao deslocamento necessário no eixo  $z$ . Além disso, a função de custo de um estado  $(x,y,z)$  do robô depende da distância da posição para o obstáculo vertical mais próximo do mapa.

Uma vez que temos a trajetória 2,5-D calculada com D\* lite, nós a aumentamos com a componente  $\psi$ . Uma vez que o scanner a laser está se dirigindo para a frente, é desejável que o robô vire para a direção referente ao primeiro voo para evitar colisões. Por outro lado, queremos que o quadroto realize pequenas manobras, como voar 10 centímetros para trás, sem se virar imediatamente. Para alcançar este comportamento, calculamos o ângulo desejado que resultaria em voar para a frente de acordo com a estrutura do quadroto. Evitar a rotação inicial e permitir o movimento para a célula desejada sem tal rotação, possibilita que o robô se movimente puramente para os lados ou mesmo para trás e impede o veículo de realizar manobras não naturais.

Em vez de mudar para um novo plano em cada instante, procuramos reutilizar solução existente, sempre que possível. Um novo plano é gerado apenas quando o plano real não é mais válido devido a obstáculos dinâmicos ou quando um determinado período de tempo foi alcançado ( $\Delta_t = 500ms$ ). Esta última restrição nos permite corrigir desvios na trajetória que foram introduzidos para evitar obstáculos que não estão mais presentes. Em nossa implementação, usamos uma resolução da grid de 4cm. Com essas configurações, o planejador exige cerca de 50-80ms para computar um caminho típico de 10m. Replanejamento pode ser feito em menos de 10ms. Obstáculos dinâmicos são detectados pelo planejador, considerando os pontos finais dos feixes de laser que não são explicados pelo mapa já conhecido através da subtração do cenário de fundo. Além disso, executamos um módulo de prevenção de obstáculos reativo embarcado que roda em paralelo e se baseia em campos potenciais [23].



**Fig. 5:** Localização global do nosso quadroto em um mapa previamente obtido a partir de uma plataforma terrestre. As marcações verde e azul evidenciam a estimativa atual da partícula do filtro e a pose atual, respectivamente. Partículas são mostradas como pontos pretos no espaço. (Acima) A situação inicial. (Meio e abaixo) depois de 1 e 5m de voo. No último, o quadroto foi localizado.

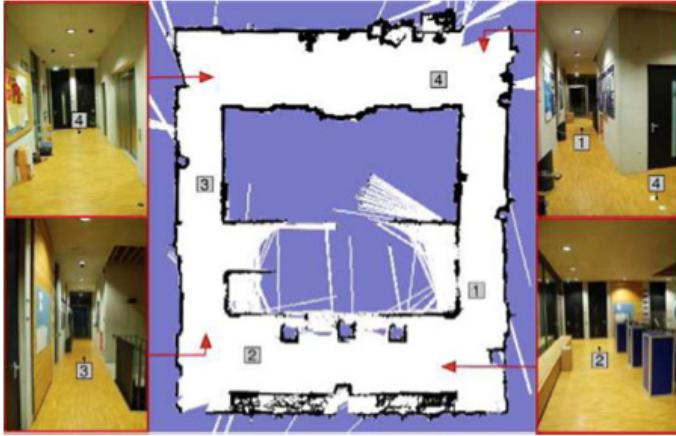
## V. EXPERIMENTOS

Nesta seção, apresentamos experimentos que mostram os desempenhos de cada um dos módulos apresentados na seção anterior, ou seja, localização, SLAM, o mapeamento multinível, estabilização da pose autônoma, planejamento de caminho e prevenção de obstáculos. Vídeos dos vários experimentos podem ser encontrados na Web [5].

### A. Localização

Utilização de mapas de grid 2-D para localização permite que o nosso sistema funcione com mapas adquiridos por diferentes tipos de robôs e não necessariamente com mapas construídos pelo próprio veículo voador. Nesta seção, apresentamos um experimento no qual realizamos a localização global do quadroto voando em um mapa adquirido com um robô terrestre. Este robô foi equipado com um scanner a laser Sick LMS. A altura do scanner era de 80 cm. Ao longo deste experimento, o veículo aéreo não tripulado (VANT) manteve uma altura de  $50 \pm 10$  centímetros e o algoritmo de filtro de partículas empregou 5.000 partículas. Dado este número de partículas, a nossa implementação atual requer 5ms por iteração em um laptop Dual-Core 2GHz, enquanto a varredura requer 5ms em média. **Fig. 5** mostra três fotos instantâneas do processo de localização em três momentos diferentes. A imagem superior representa a situação inicial, na qual as partículas foram amostradas uniformemente sobre o espaço livre. Depois de cerca de 1 metro de voo (imagem do meio), as partículas começam a se concentrar em torno da verdadeira pose do veículo. Depois de aproximadamente 5 metros de voo, o

quadrotor foi globalmente localizado (imagem de baixo). O círculo azul indica a estimativa atual do filtro.



**Fig. 6:** Mapa de um escritório construído com a nossa abordagem usando o quadrotor. Os rótulos 1-4 refletem a localização de marcações individuais que são usadas para avaliar a acurácia da nossa abordagem de mapeamento. Triângulos vermelhos indicam as posições das imagens das câmeras correspondentes. A imperfeição na parte de baixo do mapa é originada a partir do banco que contém divisões horizontais (veja imagem inferior à direita)

### B. SLAM

Também avaliamos o sistema de mapeamento, permitindo o quadrotor realizar quatro circuitos (cerca de 41m cada) em um prédio em forma de retângulo com corredor de tamanho aproximado de  $10\text{m} \times 12\text{m}$ . O resultado do nosso algoritmo SLAM é mostrado na **Fig. 6**. Para avaliar quantitativamente a precisão do nosso sistema de mapeamento, colocamos marcadores no chão (rotulados de 1 à 4) e manualmente pousamos o quadrotor perto dos marcadores. Uma vez que nunca pousamos perfeitamente sobre as marcas, nós mudamos manualmente o quadrotor os centímetros restantes para coincidir com os lugares pré-definidos. Isso nos permite medir três tipos de erros: o erro de relocalização, o erro de posicionamento absoluto, e o erro em malha aberta. O erro de relocalização é a diferença entre a estimativa atual e a estimativa da pose real no circuito anterior. O erro em malha aberta é o erro de relocalização sem possibilitar a otimização de gráfico. O erro absoluto é a diferença entre a estimativa da pose e a verdade terrestre. Para medir o erro absoluto medimos manualmente as posições relativas dos marcadores e as compararmos com as posições estimadas pelo robô ao pousar em direção aos marcadores correspondentes. A **Tabela 2** mostra as posições  $xy$  medidas manualmente e as poses estimadas dos marcadores para todos os circuitos. Como pode ser visto, tanto o erro relativo entre os circuitos individuais e as estimativas de pose globais em relação ao chão têm um erro máximo de 1 cm. Neste experimento, o mapeamento incremental durante o primeiro circuito foi suficientemente

preciso (menor que 1cm de erro); portanto, não foram necessárias otimizações já que todos os circuitos subsequentes também foram relocalizações do mapa existente. Nós também avaliamos cada circuito independentemente uns dos outros sem a otimização através de grafo. Os resultados dos circuitos de voos individuais para o marcador 4 (origem) estão apresentados na **Tabela 3** (primeira linha). O pior voo (segundo circuito), resultou em um erro de cerca de 0,37 m de distância total para a origem. As linhas restantes da **Tabela 3** mostram o efeito do uso de diferentes resoluções de grid em um nível mais detalhado da nossa abordagem de mapeamento hierárquico na acurácia dos circuitos individuais.

**Tabela 2:** Localizações estimadas e manualmente medidas dos marcadores para o voo contendo quatro circuitos no total

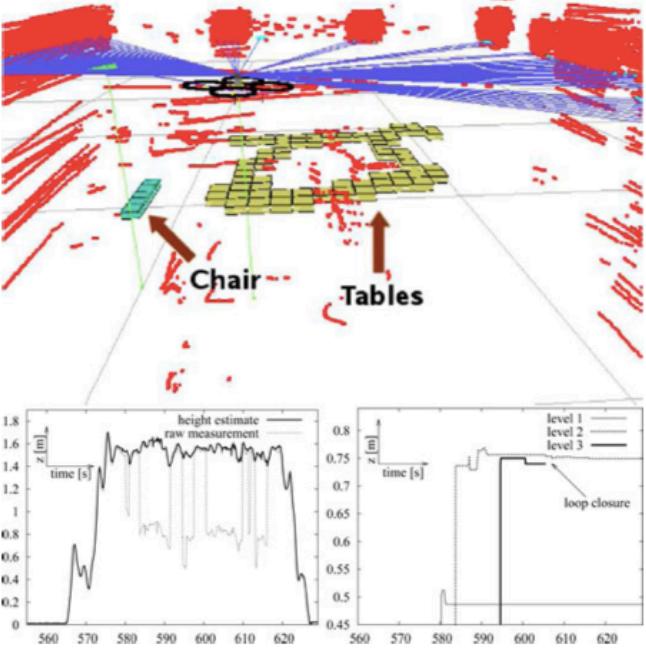
marcador	Círcuito 1	Círcuito 2	Círcuito 3	Círcuito 4	Valor real
$x_1$	<b>1.11m</b>	<b>1.11m</b>	<b>1.11m</b>	<b>1.10m</b>	<b>1.11m</b>
$y_1$	-7.50m	-7.51m	-7.50m	-7.50m	-7.50m
$x_2$	-6.21m	-6.21m	-6.21m	-6.21m	-6.21m
$y_2$	-9.21m	-9.21m	-9.21m	-9.21m	-9.21m
$x_3$	-7.85m	-7.85m	-7.85m	-7.85m	-7.85m
$y_3$	-3.83m	-3.83m	-3.83m	-3.82m	-3.82m
$x_4$	-0.01m	-0.01m	-0.01m	-0.01m	0.00m
$y_4$	-0.00m	-0.00m	-0.00m	-0.00m	0.00m

**Tabela 3:** Comparação de um único circuito para diferentes resoluções da grid

marcador	Círcuito 1	Círcuito 2	Círcuito 3	Círcuito 4	Maior resolução
$x_4$	-0.01m	-0.35m	-0.08m	-0.17m	<b>0.01m</b>
$y_4$	-0.00m	0.12m	-0.07m	0.04m	
$x_4$	-0.42m	-0.59m	-0.36m	-0.64m	<b>0.02m</b>
$y_4$	0.20m	0.23m	0.11m	0.33m	
$x_4$	-0.91m	-0.59m	-0.54m	-0.60m	<b>0.04m</b>
$y_4$	0.28m	0.38m	0.29m	0.29m	

### C. SLAM multinível e estimação da altitude

A seguir, mostramos o comportamento típico do nosso módulo para estimar a altitude. Neste experimento, deixamos o robô voar autonomamente em um escritório típico contendo cadeiras, mesas, e muita ninharia. As cadeiras têm uma altura de 48 cm, e as mesas estão dispostas próximasumas das outras, com uma altura de 77 cm. Durante esta missão, o sistema voou uma vez sobre a cadeira e várias vezes ao longo das mesas sobre as quais também voavam em um loop. **Fig. 7** mostra uma foto instantânea do nosso sistema de mapeamento multinível durante esta missão. Como pode ser visto a partir desta figura, o nosso algoritmo detectou corretamente os objetos e seus níveis correspondentes. As alturas estimadas da cadeira e as mesas foram  $48.6 \pm 2.7\text{cm}$  e  $74.9 \pm 2.8\text{cm}$ , respectivamente.



**Fig. 7:** Estimativa da altura global do veículo e o nível do chão abaixo. Quando o quadrotor se move sobre um novo nível, o laser indica uma transição entre níveis. A estimativa da altura de cada nível é refinada quando o veículo retorna para um nível particular. (Acima) O escritório. (Meio) Mapa correspondente depois de um voo autônomo sobre objetos verticais com uma altitude desejada de 150cm. (Abaixo à esquerda) Gráfico mostrando a altura estimada do veículo no tempo *versus* as medidas manuais. As alturas dos níveis correspondentes estão mostradas no gráfico abaixo à direita. Note que o Nível 3 é mesclado ao Nível 2 ao final do circuito.

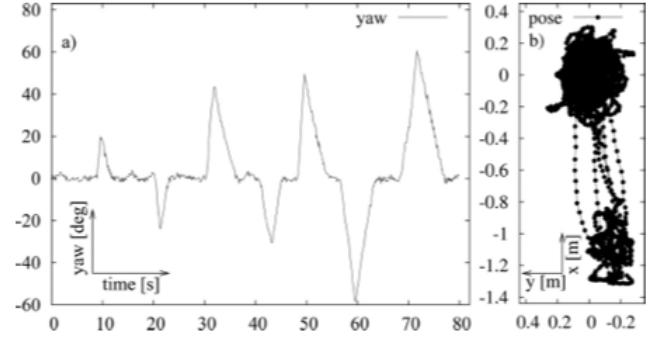
#### D. Controle da Pose

Uma vez que o sistema é estabilizado por controladores independentes, discutimos o resultado de cada controlador individualmente.

1) Controle de guinada: Para testar o controlador de guinada, definimos a guinada desejada para  $0^\circ$ , e de vez em quando, viramos o helicóptero através do controle remoto. Quando o usuário liberou o controle remoto, o veículo sempre voltou para a guinada desejada com um erro de  $\pm 2^\circ$ . **Fig. 8 (a)** representa graficamente os resultados de um ensaio típico para a estabilização de guinada.

2) Controle de Altitude: De forma semelhante ao experimento para guinada, fizemos um experimento para avaliar o comportamento do controle de altitude. Neste teste definimos a altitude desejada para 150cm. No início, o veículo estava pairando sobre o chão. Depois de ativar a estabilização, o veículo começou a subir para a altitude desejada. A altura desejada foi mantida pelo veículo com um erro máximo variando de  $\pm 10\text{cm}$ . Os resultados são mostrados da **Fig. 7**. Note que este experimento foi realizado ao voar sobre diferentes elevações.

3) Controle x,y: Finalmente mostramos um experimento apenas para a estabilização. Note que a estabilidade da pose é fortemente afetada pela latência do sistema (ou seja, o tempo necessário para calcular o comando, uma vez obtidos os dados do laser). Embora as estimativas de movimento incrementais demorem apenas 5ms em média (com um máximo de 15ms), temos que lidar com uma latência de 120ms em média, devido à transmissão sem fio e ao buffer do sensor. Uma corrida típica incluindo estabilização de pose autónoma é mostrada da **Fig. 8(b)**. Aqui, o quadrotor foi configurado para manter a pose inicial de  $(0,0)$ , e de vez em quando, o usuário usou o controle remoto para movimentar a quadrotor a cerca de 1m para trás. O quadrotor então autonomamente voltou para a posição desejada. Dependendo da latência no sistema, as oscilações de pose são tipicamente cerca de  $\pm 20\text{cm}$ .

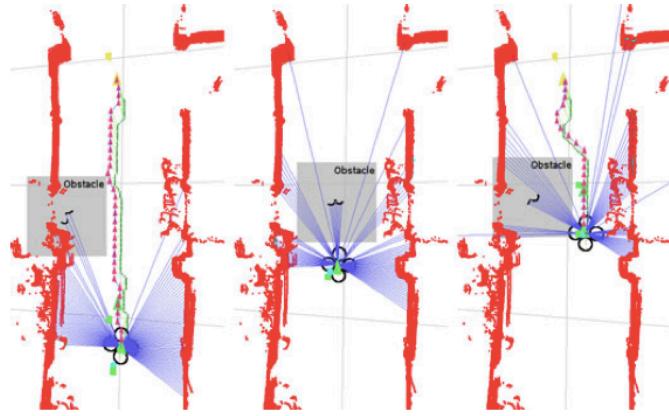


**Fig. 8:** Experimentos para a estabilização autônoma da guinada(a) e pose(b). Durante o experimento de estabilização da guinada, o quadrotor foi requerido para rodar para  $0^\circ$ , enquanto o homem manualmente rodou o robô para uma posição aleatória. No experimento de estabilidade da pose, o quadrotor foi configurado para pairar a  $(0,0)$ , mas foi manualmente movido para trás, o que exigiu que o quadrotor retornasse para sua pose inicial.

#### E. Planejamento de trajetória e desvio de obstáculos

Nesta seção, apresentamos um experimento que demonstra nossos algoritmos de planejamento de trajetória e desvio de obstáculos dinâmicos. Ao quadrotor, foi dado um ponto objetivo de cerca de 5m na sua frente. Uma pessoa estava em pé ao lado esquerdo (veja a área sombreada da **Fig. 9**) e entra no corredor enquanto o quadrotor se movia para o objetivo desejado. A segunda imagem mostra a situação em que a pessoa está completamente bloqueando o caminho do robô. Neste caso, o quadrotor girava em torno do último ponto de forma válida, uma vez que não havia mais um plano válido para o objetivo. Quando a pessoa se moveu para a esquerda novamente, o quadrotor foi capaz de realizar um desvio, como mostrado na imagem direita da **Fig. 9**. Note que as fotos instantâneas mostram apenas os pontos finais do laser. Embora

pareça que o quadrotor pode ter espaço para voar ao redor da pessoa na segunda imagem, não há um plano válido por causa das margens de segurança ao redor das paredes.



**Fig. 9:** Experimento de planejamento de trajetória e desvio de obstáculos dinâmicos. Ao quadrotor é dado um destino a 5m a sua frente. A trajetória planejada é mostrada na imagem à esquerda. Uma pessoa entra no corredor (área sombreada) e bloqueia o caminho do robô, que resulta em um plano inválido. O quadrotor, então pula ao redor do último ponto válido (segunda imagem). Na terceira imagem, a pessoa sai do caminho, permitindo que o quadrotor retome sua trajetória.

## VI. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, um sistema de navegação para o voo indoor autônomo, utilizando uma plataforma aberta de hardware para um quadrotor, foi apresentado. Uma solução de navegação completa que aborda aspectos diferentes de localização, mapeamento, planejamento de rota, estimativa de altitude e controle foi descrita. Uma vez que não conta com características específicas de plataformas de voo, como modelo de dinâmica, acreditamos que nosso sistema possa ser facilmente adaptado à diferentes veículos voadores. Todos os módulos do nosso sistema funcionam em tempo real. No entanto, por causa do custo computacional relativamente alto de alguns algoritmos, apenas uma parte do software é executada no processador ARM embarcado, enquanto a outra parte é executada em um notebook. Alguns testes preliminares nos deixam confiantes de que todo o sistema possa funcionar embarcado com o uso da próxima geração de computadores embutidos, que são baseados no processador Intel Atom. Nós fornecemos uma ampla gama de experiências e alguns vídeos que destacam a eficácia do nosso sistema. Em trabalhos futuros, pretendemos adicionar dados de câmera no nosso sistema. Acredita-se que esta tecnologia possa ser efetivamente integrada e vai nos permitir relaxar a suposição de que o veículo se move sobre uma superfície plana por partes.

## VII. REFERÊNCIAS

- [1] Carmen. (Oct. 8, 2011). [Online]. Disponível: <http://carmen.sourceforge.net/>
- [2] ROS—Robot Open Source. (Oct. 8, 2011). [Online]. Disponível: <http://www.ros.org>
- [3] Mikroopter.(Oct.8,2011).[Online].Disponível:<http://www.mikroopter.de/>
- [4] OpenSLAM—Open Source Navigation Software Repository, (Oct. 8, 2011). [Online]. Disponível: <http://www.openslam.org>
- [5] (Oct. 8, 2011). [Online]. Disponível: <http://ais.informatik.uni-freiburg.de/projects/quadrotor/>
- [6] M.Achtelika, A.Bachrach, R.He, S.Prentice, and N.Roy, “Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments,” em Proc. Robot.: Sci. Syst., 2008.
- [7] S.Ahrens,D.Levine,G.Andrews, and J.P.How, “Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments,” em Proc. IEEE Int. Conf. Robot. Autom., 2009, pp. 2643–2648.
- [8] E.Altug, J. P. Ostrowski, and R. Mahony, “Control of a quadrotor helicopter using visual feedback,” em Proc. IEEE Int. Conf. Robot. Autom., 2002.
- [9] A. Bachrach, R. He, and N. Roy, “Autonomous flight in unknown indoor environments,” Int. J. Micro Air Veh., vol. 1, no. 4, 2009.
- [10] S.Bouabdallah,C.Bermes,S.Gronzka,C.Gimkiewicz,A.Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart, “Towards palm-size autonomous helicopters,” presented at the Int. Conf. Exhib. Unmanned Areal Veh., Dubai, UAE, 2010.
- [11] S. Bouabdallah and R. Siegwart, “Full control of a quadrotor,” em Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2007.
- [12] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, “Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle,” IEEE Trans. Robot., vol. 25, no. 3, pp. 743–749, Jun. 2009.
- [13] K. Celik, S. J. Chung, M. Clausman, and A. K. Soman, “Monocular vision SLAM for indoor aerial vehicles,” em Proc. IEEE Intell. Robots Syst., 2009, pp. 1566–1573.
- [14] A. Coates, P. Abbeel, and A. Y. Ng, “Learning for control from multiple demonstrations,” em Proc. Int. Conf. Mach. Learning, 2008.
- [15] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” presented at the IEEE Int. Conf. Robot. Autom., Leuven, Belgium, 1998.
- [16] G. Grisetti, C. Stachniss, and W. Burgard, “Non-linear constraint network optimization for efficient map learning,” IEEE Trans. Intell. Transp. Syst., vol. 10, pp. 428–439, Sep. 2009.
- [17] S. Grzonka, G. Grisetti, and W. Burgard, “Towards a navigation system for autonomous indoor flying,” em Proc. IEEE Int. Conf. Robot. Autom., Kobe, Japan, 2009.
- [18] R. He, S. Prentice, and N. Roy, “Planning in information space for a quadrotor helicopter in a GPS-denied environment,” em Proc. IEEE Int. Conf. Robot. Autom., 2008.
- [19] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, “The Stanford testbed of autonomous rotorcraft for multiagent control,” em Proc. 23rd Digital Avion. Syst. Conf., 2004, p. 2.
- [20] N. G. Johnson, “Vision-assisted control of a hovering air vehicle in an indoor setting,” Master’s thesis, Dept. Mech. Eng., Brigham Young Univ., Provo, UT, 2008.
- [21] C. Kemp, “Visual control of a miniature quad-rotor helicopter,” Ph.D. dissertation, Churchill College Univ., Cambridge, U.K., 2005.
- [22] S. Koenig and M. Likhachev, “Fast replanning for navigation in unknown terrain,” IEEE Trans. Robot., vol. 21, no. 3, pp. 354–363, Jun. 2005.
- [23] J.-C. Latombe, Robot Motion Planning. Norwell, MA: Kluwer, 1991, pp. 295–356.
- [24] E. Olson, “Real-time correlative scan matching,” em Proc. IEEE Int. Conf. Robot. Autom., Kobe, Japan, Jun. 2009, pp. 4387–4393.
- [25] P. Pounds, R. Mahony, and P. Corke, “Modelling and control of a quadrotor robot,” em Proc. Australian Conf. Robot. Autom., 2006.
- [26] J. F. Roberts, T. Stirling, J. C. Zufferey, and D. Floreano, “Quadrotor using minimal sensing for autonomous indoor flight,” em Proc. Eur. Micro Air Veh. Conf. Flight Competition, 2007.
- [27] N.Roy,M.Montemerlo, and S.Thrun, “Perspectives on standardization in mobile robot programming,” presented at the IEEE/RSJ Int. Conf. Intell. Robots Syst., Las Vegas, NV, 2003..

- [28] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *Int. J. Robot. Res.*, vol. 27, no. 5, p. 549, 2008.
- [29] A. Tayebi and S. McGilvray, "Attitude stabilization of a VTOL quadrotor aircraft," *IEEE TX Control Syst. Technol.*, vol. 14, no. 3, pp. 562–571, May 2006.
- [30] T. Templeton, D. H. Shim, C. Geyer, and S. S. Sastry, "Autonomous vision-based landing and terrain mapping using an MPC-controlled unmanned rotorcraft," *em Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 1349–1356.
- [31] S. Thrun, W. Burgard, and D. Fox, "Robot perception," *em Probabilistic Robotics*, Cambridge, MA: MIT Press, 2005, pp. 171–172.
- [32] S. Thrun, M. Diel, and D. Hahnel, "Scan alignment and 3-D surface modeling with a helicopter platform," *Field Serv. Robot. (STAR Springer Tracts Adv. Robot.)*, vol. 24, pp. 287–297, 2006.
- [33] G. P. Tournier, M. Valenti, J. P. How, and E. Feron, "Estimation and control of a quadrotor vehicle using monocular vision and moire patterns," *em Proc. AIAA Guid., Navigat. Control Conf. Exhibit.*, 2006, pp. 21–24.